VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA

EKONOMICKÁ FAKULTA

KATEDRA SYSTÉMOVÉHO INŽENÝRSTVÍ

Programový modul pro uživatelský návrh tiskových výstupů

Development of a Custom Reports Definition Module

Student: Bc. Radim Mikula

Vedoucí bakalářské práce: RNDr. Ivo Martiník, Ph.D.

Ostrava 2015

VŠB - Technická univerzita Ostrava
Ekonomická fakulta
Katedra aplikované informatiky

Zadání diplomové práce

Student: **Bc. Radim Mikula**

Studijní program: N6209 Systémové inženýrství a informatika

Studijní obor: 6209T025 Systémové inženýrství a informatika

Téma: Programový modul pro uživatelský návrh tiskových výstupů
Development of a Custom Reports Definition Module

Zásady pro vypracování:

1. Úvod
2. Metodologická východiska práce v oblasti technologií Java
3. Analýza business požadavků
4. Návrh koncepce řešení pro vývoj programového modulu
5. Použití certifikátů k odesílání šifrovaných zpráv
6. Zhodnocení navržené koncepce a implementace programového modulu
7. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků diplomové práce
Seznam příloh
Přílohy

Seznam doporučené odborné literatury:

DASHORST, Martijn and Eelco HILLENIUS. *Wicket in action.* Greenwich: Manning, 2009. 392 p. ISBN 19-323-9498-2.
KONDA, Madhusudhan. *Just Hibernate.* Aufl: O'Reilly Media, 2014. 140 p. ISBN 978-144-9334-376.
SCHAEFER, Chris, Clarence, HO and Rob HARROP. *Pro Spring.* London: Apress, 2014. 728 p. ISBN 978-143-0261-513.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **RNDr. Ivo Martiník, Ph.D.**

Datum zadání: 21.11.2014
Datum odevzdání: 25.04.2015

Ing. Petr Rozehnal, Ph.D.
vedoucí katedry

prof. Dr. Ing. Dana Dluhošová
děkanka fakulty

Prohlašuji, že jsem celou práci, včetně všech příloh, vypracoval samostatně.

……

Bc. Radim Mikula

V Ostravě dne 25.04.2015

Tímto bych chtěl poděkovat RNDr. Ivo Martiníkovi, Ph.D. za cenné rady a odbornou pomoc při vedení této diplomové práce.

Bc. Radim Mikula

Obsah:

1 Úvod 5

2 Metodologická východiska práce v oblasti technologií Java 7

2.1 Softwarové inženýrství 7

2.1.1 Vodopádový model 7

2.2 Unified Modeling Language 8

2.2.1 Objekty a jazyk UML 9

2.2.2 Struktura jazyka UML 9

2.2.3 Společné mechanismy jazyka UML 13

2.2.4 Architektura 15

Modelování případů užití 15

2.3 Základní pojmy 17

2.3.1 Internet 17

2.3.2 Webová aplikace 17

2.4 Technologie 19

2.4.1 JasperReports 19

2.4.2 Spring Framework 21

2.4.3 Hibernate 23

2.4.4 Google Web Toolkit 24

2.4.5 Sencha GXT 28

2.5 Nástroje pro implementaci 29

2.5.1 Definice projektů pomocí Maven 29

3 Analýza business požadavků 30

3.1 Specifikace softwarových požadavků 31

3.2 Atributy požadavků 31

3.3 Shrnutí a analýza požadavků společnosti u&sluno 32

3.3.1 Popis a rozdělení požadavků 32

R6 - Ukládání tiskové sestavy do souboru 34

3.3.2 Návrh případu užití 36

3.3.3 RTM matice sledovatelnosti požadavků 36

3.3.4 Use case diagram 37

4 Návrh koncepce řešení pro vývoj programového modulu 39

4.1 Architektura aplikace ... 39
4.1.1 DATA ACCESS OBJECT (DAO) ... 41
4.1.2 DATA TRANSFER OBJECT (DTO) ... 41
4.2 Struktura aplikace ... 42
4.3 Ukázky implementace aplikace ... 44
4.3.1 GWT konfigurace ... 44
4.3.2 Hibernate konfigurace ... 45
4.3.3 Spring Framework konfigurace ... 45
4.3.4 Implementace návrhového vzoru DAO ... 46
4.3.5 Implementace návrhového vzoru DTO ... 47
4.3.6 ER diagram datového modelu ... 48
4.4 Vzhled uživatelského rozhraní aplikace ... 50
4.4.1 Správa tiskových šablon ... 50
4.4.2 Výběr dat ... 52
4.4.3 Navigátor ... 53
4.4.4 Prezentace dat ... 54
4.4.5 Kontrolor ... 55
4.4.6 Chybová Konzole ... 57
4.4.7 Správa obrázků ... 57
4.5 Sekvenční diagram pro generování tiskového výstupu ... 58
5 Použití certifikátů k odesílání šifrovaných zpráv ... 59
6 Zhodnocení navržené koncepce ... 60
6.1 Zhodnocení etapy realizace webové aplikace ... 61
7 ZÁVĚR ... 63
Seznam použité literatury ... 65
Seznam zkratek ... 67
Seznam obrázků ... 69
Seznam tabulek ... 70
Prohlášení o využití výsledků diplomové práce ... 3
Prohlašuji, že ... 3
Seznam příloh ... 4

1 Úvod

Informační systémy v dnešní době zpracovávají obrovské množství dat. Tato data jsou především zpracovávána počítačově, protože jiné formy prakticky nejsou realizovatelné. Ovšem úlohou běžného informačního systému[1] není jen zpracovávání dat, ale také poskytování zpracovaných dat ve vhodné grafické formě jeho uživatelům.

Vhodná prezentace dat se může velmi lišit a to zejména množstvím prezentovaných dat, jejich strukturou a charakterem. Je tedy nutné prezentované data tomu přizpůsobit. Osvědčilo se jednotlivé oddíly vhodně odlišit písmem a barvou. Větší množství dat vhodně rozdělit stránkováním a případně využít grafy pro snadnější a rychlejší orientaci. Tyto pokročilé výstupy jsou označovány jako „tiskové sestavy“, zejména z historického hlediska, kdy byly především tištěny na papír.

Pokročilejší informační systémy mohou obsahovat připravené konkrétní tiskové sestavy, ale také možnosti vytváření nových sestav, které jsou uživatelem navržené.

Nástroj sloužící k přípravě vlastních tiskových sestav, je označován jako „návrhář tiskových sestav“. Uživatel pak může tento nástroj využívat pro svůj vlastní výběr dat a vlastní formu prezentace. Tyto funkce jsou pro uživatele klíčové. K výběru dat můžeme použít agregační funkce nebo omezující podmínky. Vytvořená prezentace může být kombinací prostých seznamů, detailních tabulek, různých typů grafů a dalších forem prezentace, jako jsou čárové kódy a obrázky.

Cílem této práce je provést analýzu, návrh a implementovat programový modul pro uživatelský návrh tiskových výstupů. Tento modul bude sloužit jako samostatná webová aplikace, která může komunikovat s ostatními informačními systémy pomocí webových služeb.

[1] Výraz systém, software, aplikace, produkt budou v této diplomové práci volně zaměňovány. Probírané principy a postupy se vztahují na libovolné programy a systémy se softwarem.

Cíl práce:

1. Shrnout teoretické poznatky a přiblížit metody vývoje webové aplikace, dále zde nalezneme studii a popis technologií Jasper Reports, Google Web Toolkit a dalších souvisejících technologií nutných k implementaci potřebného softwaru,
2. Specifikovat uživatelské požadavky kladené na webovou aplikaci a na jejich základě analyzovat a navrhnout potřebné řešení,
3. provedení implementace aplikace a její zhodnocení.

První část práce je zaměřena na základní popis technologií spojených s moderními nástroji pro vytváření webových aplikací. Rovněž zde nalezneme veškeré nástroje a postupy, které byly použity při tvorbě webové aplikace.

V druhé části nalezneme analýzu businness požadavků. Jsou zde veškeré požadavky na webovou aplikaci a analytický návrh ve formě diagramů pro budoucí webovou aplikaci.

Poslední část popisuje potřebné kroky implementace webové aplikace. Základní logiku a strukturu aplikace.

Cílem práce není popsat možnosti všech technologií, metodik nebo návrhových vzorů při vývoji webové aplikace. Vzhledem rozsahu práce to není ani možné. Předpokládáme, že čtenář má určité znalosti v oblasti objektově-orientovaného programování, analýzy a návrhu softwaru, databázového zpracování. Dále jsou přepokládány znalosti jazyků JAVA a technologií HTML a CSS. V první polovině práce bude věnována pozornost teoretické části, která se zabývá oborem softwarového inženýrství a metodik vývoje softwaru obecně. Dále zde nalezneme technologie použité pro vývoj webové aplikace, jehož implementace je jedním z cílů práce. Druhá polovina práce se zabývá analýzou, návrhem a implementací webové aplikace.

2 Metodologická východiska práce v oblasti technologií Java

V této kapitole nalezneme veškeré teoretické podklady této práce a standardizovaný postup návrhu a implementace programového modulu. Její přečtení má čtenáři přiblížit myšlenkové postupy, které autor použil při vývoji softwarové aplikace.

Nejprve zde nalezneme obecné postupy, myšlení a způsoby přístupu softwarového inženýrství a veškeré technologie pro tvorbu programového modulu tiskových výstupů. Není ani zapomenut jazyk UML, který zajišťuje unifikovaný proces vývoje aplikace.

2.1 Softwarové inženýrství

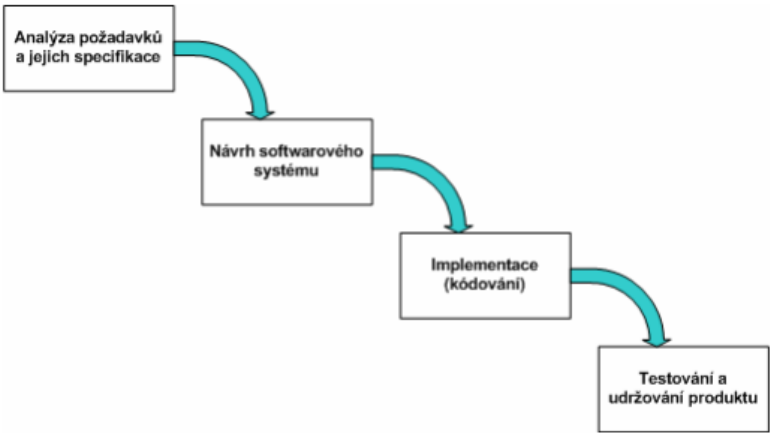
Softwarové inženýrství je definováno podle (Stephens, 2015) jako: *"Organizovaný, analytický přístup k návrhu, vývoje, používání a údržbu softwaru."*

Více intuitivní popis softwarového inženýrství je zavedení a používání řádných principů, abychom dosáhli ekonomické tvorby softwaru, který je spolehlivý a pracuje účinně na dostupných výpočetních prostředích. (Fritz Buer, 1968). Software může být dále rozšiřován, aby uspokojil měnicí potřeby zákazníků pro nadcházející roky.

Mezi hlavní přínosy softwarového inženýrství patří metodiky a životní cyklus aplikací. Stejně jako v jiných odvětvích i tento obor procházel určitým vývojem. Vždy bylo snahou přizpůsobit vývoj softwaru aktuálním požadavkům, které byly na software kladeny v dané době. S podmínkou, že s každým přizpůsobením bylo žádoucí opravit nedostatky, které se objevovaly v předchozích postupech (Kadlec, 2004). Vodopádový model je základním modelem pro vývoj softwaru, který bude dále popsán.

2.1.1 Vodopádový model

Tento model byl vyvinut Dr. Winstonem v roce 1970. V době vzniku tento model představoval naprostou revoluci. Model vychází z posloupnosti přímočarém sledu fází a navíc umožňuje návrat na předchozí fázi. Dodnes neexistuje přesně definovaná podoba softwarového procesu, který můžeme označit jako referenční. Nicméně lze říci, že vodopádový model se stál základem, který je možno nalézt v různých modifikacích a rozšířeních ve většině současných přístupů. Na obrázku číslo 1. můžeme vidět v grafickém znázornění vodopádový model. (Kadlec, 2004).



Obrázek 1: Schéma vodopádového modelu (Zdroj: Vondrák, 2002)

Princip vodopádového modelu spočívá v tom, že následující množina činností spojená s danou fází nemůže započít dříve, než skončí předchozí. Nedostatky tohoto základního modelu softwarového procesu, které vedly k jeho modifikacím, jsou podle (Vondrák, 2002) následující:

- příliš dlouhá prodleva mezí zadáním projektu a vytvořením spustitelného softwaru,
- výsledek záleží na přesném zadání požadavků na výsledný produkt,
- nelze odhalit výslednou kvalitu softwaru, dokud není výsledný softwarový systém hotov.

Snaha o odstranění všech nedostatků vedla k různým modifikacím. Například se jedná o model *inkrementální* postavený na principu vytváření verzí, které zahrnují širší škálu funkcí definovaných v průběhu vytváření softwarového systému. V podstatě se jedná o řadu menších vodopádových modelů s výrazně kratším životním cyklem, kde každý odpovídá nové sadě doplňujících požadavků. (Vondrák, 2002)

Pro návrh softwarových systému v softwarovém inženýrství se používá jazyk UML, který nalezneme v další podkapitole.

2.2 Unified Modeling Language

Jazyk UML (Unified Modeling Language) a jeho nástroje budou v této práci použity pro popis komponent návrhu webové aplikace.

Je to univerzální jazyk pro vizuální modelování. *„Tento jazyk byl navržen proto, aby spojil nejlepší existující postupy modelovacích technik a softwarového inženýrství. Diagramy*

vytvořené v jazyku UML jsou srozumitelné pro lidi, ale navíc je mohou snadno interpretovat i programy CASE (computer-aided sotware engineering).“ (Arlow & Neustadt, 2007, str. 28)

UML se v současnosti používá jako standart pro analýzu, návrh, specifikaci a dokumentaci softwarových systémů. Díky bohaté syntaxi UML podporuje objektově-orientovaný přístup k modelování. Musíme poznamenat, že UML není metodika, neobsahuje konkrétní návody, jak postupovat při vývoji. Naproti tomu slouží od prvotního konceptu až po detailní návrh softwaru. K tomu využívá velké množství diagramů, které jsou rozděleny podle povahy na strukturní diagramy, diagramy chování a diagramy interakce. (Arlow & Neustadt, 2007)

2.2.1 Objekty a jazyk UML

Jazyk UML umožňuje modelování softwaru jako „kolekce spolupracujících objektů“. Přestože tato představa zapadá do modelu objektově orientovaných softwarových systémů a programovacích jazyků, funguje stejně spolehlivě v podnikatelských a obchodních procesech a také v dalších aplikacích. (Arlow & Neustadt, 2007)

Modely UML mohou mít dva aspekty:

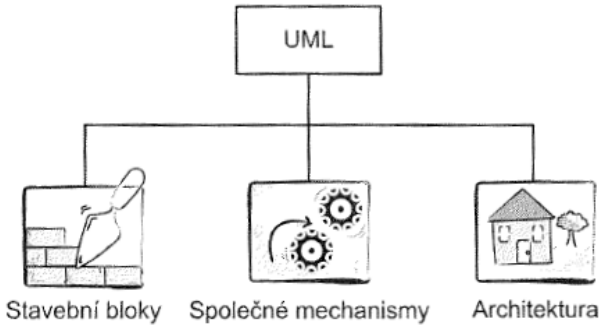
- **Statická struktura** – popisuje, jednak jaké typy objektů jsou důležité pro modelování daného systému, jednak jak spolu tyto objekty souvisejí.
- **Dynamické chování** – popisuje životní cyklus zmiňovaných objektů. Dále způsob jejich vzájemné spolupráce k dosažení požadované funkčnosti navrhovaného systému. (Arlow & Neustadt, 2007)

Pro pochopení funkce jazyka UML jako jazyka vizuálního, se musíme zaměřit na jeho strukturu.

2.2.2 Struktura jazyka UML

Struktura je znázorněna na obrázku 2 a obsahuje tyto součásti:

- ***„Stavební bloky*** *– jsou to základní prvky modelu, relace a diagramy.*
- ***Společné mechanismy*** *– obecné způsoby, jimiž se jazyku UML dosáhne specifických cílů.*
- ***Architektura*** *– pohled v jazyku UML na architekturu navrhovaného systému.“* (Arlow & Neustadt, 2007)

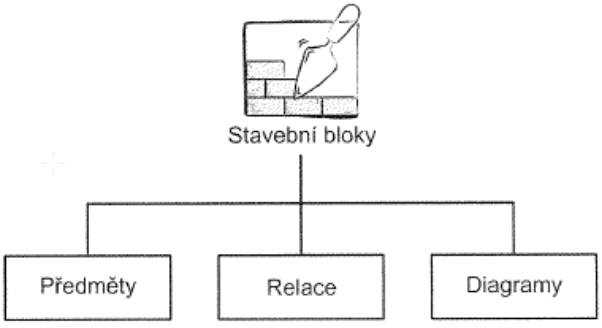


Obrázek 2: Struktura jazyka UML (Zdroj: Arlow & Neustadt, 2007)

Stavební bloky jazyka UML

Podle knihy *UML 2 a unifikovaný proces vývoje aplikací* (Arlow & Neustadt, 2007) je jazyk UML sestaven ze tří stavebních bloků:

- ***Předměty*** *(things), neboli samotné prvky modelu,*
- ***Relace*** *(relationships), jenž určují, jak spolu dva nebo více předmětů významově souvisí,*
- ***Diagramy*** *(diagrams), což jsou pohledy na modely UML; ukazují kolekce předmětů, které „vyprávějí příběh" o softwarovém systému a jsou naším způsobem vizualizace toho, co systém bude dělat (analytické diagramy), a toho, jak to bude dělat (návrhové diagramy)."* (Arlow & Neustadt, 2007)



Obrázek 3: Stavební bloky jazyka UML (Zdroj: Arlow & Neustadt, 2007, s. 35)

Předměty

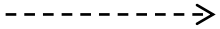
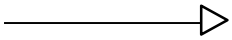
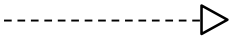
Předměty rovněž „věci" nebo abstrakce dělíme na:

- **Strukturní abstrakce**, což jsou podstatná jména modelu UML, jako jsou třídy, rozhraní, spolupráce, případ užití, aktivní třída, komponenta, uzel.
- **Chování**, což jsou slovesa modelu UML, např. interakce, stav.
- **Seskupení,** označujeme balíčky používané k seskupování sémantických souvisejících prvků modelu do spojitých jednotek.

- **Poznámky**, anotace, které můžeme k modelu připojit s úmyslem zachytit informaci vytvořenou jen k tomuto účelu (což je zvýrazněný text tak, aby poznámka v okolním textu vynikala). (Arlow & Neustadt, 2007)

Relace

Relace zachycuje v modelu vztah mezi dvěma předměty. V tabulce číslo 1 je možné vidět rozlišení následujících typů relací.

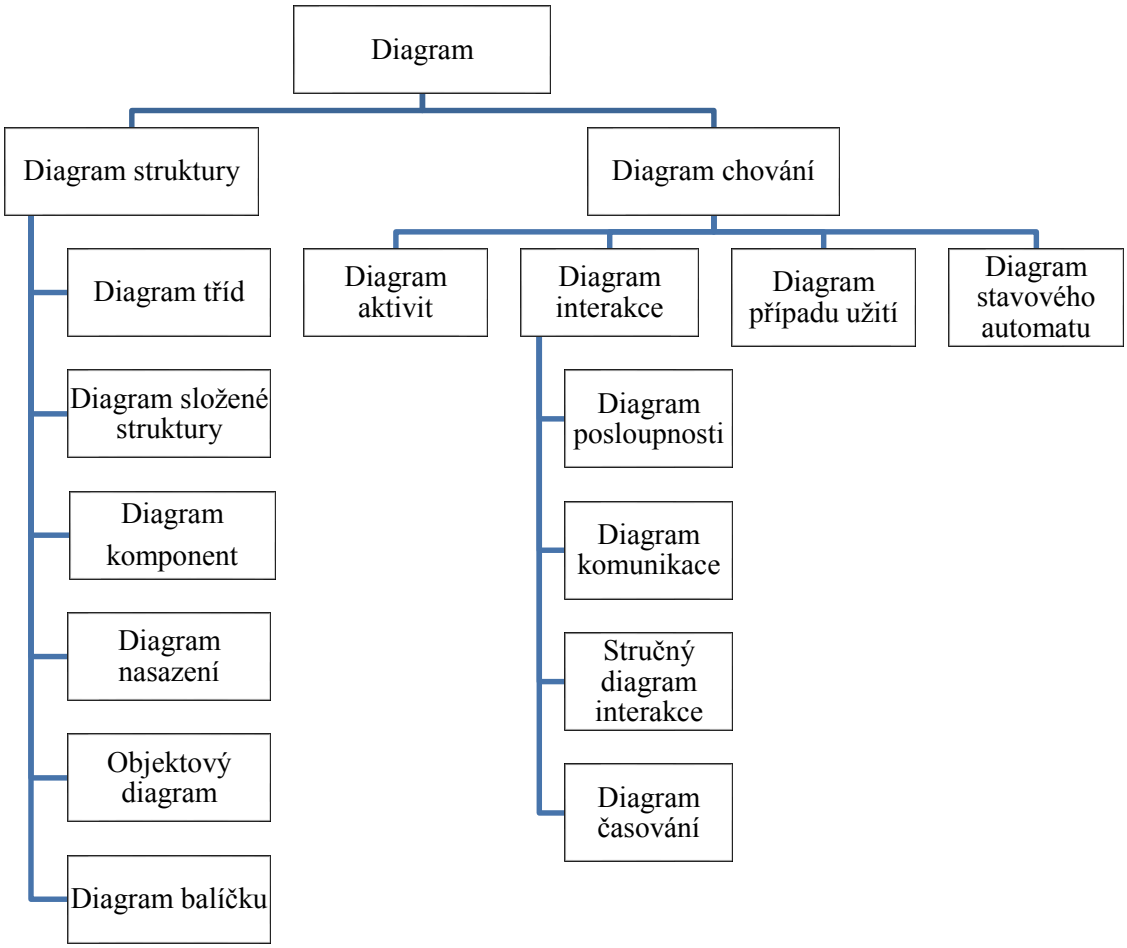
Typ relace	Syntaxe UML		Stručný popis
	zdroj	**cíl**	
Závislost (Dependency)			Změna zdrojového předmětu ovlivňuje význam závislého předmětu.
Asociace (Association)			Popis množiny spojení mezi předměty.
Agregace (Aggregation)			Zdrojový předmět je součástí cílového předmětu.
Kompozice (Compositon)			Silnější forma agregace, která má více omezení.
Ochranná nádoba (Containment)			Zdrojový předmět obsahuje cílový předmět.
Zobecnění (Generalization)			Jeden předmět je specializací jiného předmětu a lze jej nahradit obecnějším (univerzálnějším) předmětem.
Realizace (Realization)			Asociace mezi klasifikátory, kde jeden klasifikátor určuje dohodu, jejíž uskutečnění zaručuje druhý klasifikátor.

Tabulka 1: Relace jazyka UML (Zdroj: Arlow & Neustadt, 2007, s. 36)

Diagramy

Ve všech nástrojích CASE (nástroje pro tvorbu programového vybavení pomocí počítače, Computer-Aided Software Engeneering) založených na jazyku UML jsou vytvořené předměty nebo relace automaticky přidávány do vznikajícího modelu. Model obsahuje všechny předměty a relace vytvořené k tomu, aby popisovaly chování navrhovaného softwaru.

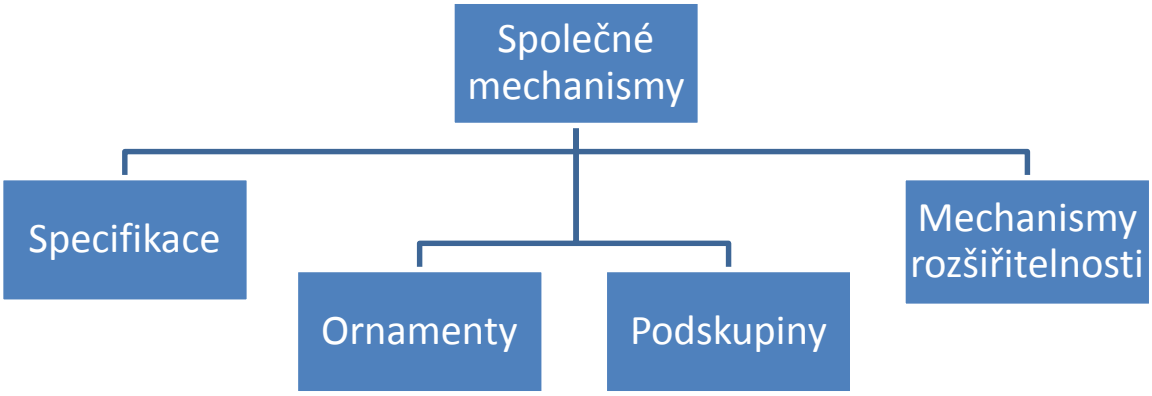
Vzhledem k tomu diagramy představují pohledy na model. V UML existuje třináct různých typu diagramů. Na obrázku 4, nalezneme veškeré UML diagramy. Můžeme vidět, že diagramy lze dělit na ty, které modelují statickou strukturu. Tyto diagramy slouží pro zobrazování jednotlivých elementů a jejich vzájemné asociace. Dále na diagramy chování, které znázorňují, jakým způsobem na sebe působí jednotlivé elementy v softwaru v čase. (Arlow, 2007) V části v návrh koncepce řešení pro vývoj programového modulu jsou použity některé výše uvedené diagramy. (Arlow, 2007)



Obrázek 4: Diagramy UML(Zdroj: Arlow & Neustadt, 2007, s. 37)

2.2.3 Společné mechanismy jazyka UML

Jazyk UML obsahuje čtyři společné mechanismy používané v celém jazyku. Tyto mechanismy popisují strategie cesty k modelování objektů, jenž jsou opakovaně používány v různých kontextech. Na obrázku číslo 5 je patrné, že jazyk UML má jednoduchou strukturu. (Arlow & Neustadt, 2007)



Obrázek 5: Společné mechanismy UML (Zdroj: Arlow & Neustadt, 2007)

Specifikace

Specifikace jsou textovým popisem jednotlivých elementů v modelu a vyjadřují jejich sémantiku. Jak tvrdí (Arlow & Neustadt, 2007) množina specifikací je „jádrem“ modelu a tvaruje „sémantické podklad, který dává smysl modelu a udržuje ho pohromadě.

Ornamenty

Ornamenty přestavují volitelné doplňky elementů, které používáme k zobrazení více informací. Slouží k vylepšování modelu. Tyto ornamenty zvyšují detailnost, na druhou stranu také složitost, a proto by se měly používat s rozvahou. Jak říká (Arlow & Neustadt, 2007): „*Musíme si uvědomit, že všechny diagramy UML jsou pouhým pohledem na daný model. Proto bychom měli používat ornamenty pouze v případě, kdy zvyšují srozumitelnost a čitelnost diagramu, případě tehdy, když zdůrazňují určitou důležitou funkci modelu. V diagramu obvykle není potřeba zobrazovat všechny podrobnosti. Mnohem důležitější je, aby byl diagram srozumitelný, aby znázorňoval přesně ty cíle, jichž chceme dosáhnout, a aby byl snadno čitelný.*“

Podskupiny

Podskupiny popisují různé způsoby vidění světa. V jazyce UML nalezneme dvě takové podskupiny. První je to skupina klasifikátorů a instancí a druhá rozhraní a implementací:

- **Klasifikátor a instance**. Klasifikátor představuje abstraktní vyjádření typu předmětu. Zatímco instance je konkrétním předmětem výskytu abstraktní představy. V UML je instance obvykle znázorněná stejným způsobem jako odpovídající klasifikátor.
- **Rozhraní a implementace**. Rozraní definuje úmluvu, které zaručuje, čím se jednotlivé implementace budou řídit. (Buchalcevová, et al., 2007)

Mechanismy rozšiřitelnosti

Jazyk UML je rozšiřitelný modelovací jazyk navržen tak, aby uspokojil potřeby všech uživatelů v současnosti i v budoucnu. Pro rozšíření můžeme využít následující mechanismy, které nalezneme v tabulce číslo 2.

Mechanismy rozšiřitelnosti	Popis
Omezení (Containts)	Omezující podmínky umožňují omezit určité chování předmětu tím, že k němu přidávají nová pravidla. Omezující podmínka je textový řetězec uzavřený do složených závorek {}, který specifikuje podmínky, které musí být vyhodnoceny pravdivě.
Stereotypy (Stereotypes)	Stereotyp představuje určitou variantu předmětu v modelu, který má sice stejnou podobu, ale používá se za jiným účelem. Tento mechanismus umožňuje vytvářet nové předměty založené na existujících. Název stereotypu se vkládá do složených závorek << >> a připojuje se k předmětu. Například <<include>>, <<extend>>, <<interface>>.
Označené hodnoty (Tagged values)	Označené hodnoty umožňují rozšířit specifikaci prvku tím, že k takovému prvku přidáme informaci sestavenou jen k tomu to účelu. Označená hodnota je dvojice prvků *jméno(tag)* a *hodnota (value)*. Obecná syntaxe vypadá takto: {tag1 = hodnota1}

Tabulka 2: Mechanismy rozšiřitelnosti jazyka UML (Zdroj: Arlow & Neustadt, 2007, s. 43)

2.2.4 Architektura

Jak tvrdí (Arlow & Neustadt, 2007) jazyk UML zachycuje strategické aspekty softwaru v architektuře „4+1“. Abychom byli schopni zachytit všechny podstatné charakteristické rysy architektury daného softwaru, definuje jazyk UML čtyři různé pohledy na systém: logický pohled, pohled procesů, pohled implementace a pohled nasazení. Všechny zmiňované pohledy jsou integrovány do pátého, který je pohled případů užití. Nyní si přiblížíme jednotlivé pohledy:

- **Logický pohled**. Zachycuje slovník oblasti problému jako množinu tříd a objektů. Důraz je kladen na zobrazení způsobu, jakým objekty a třídy implementují chovaní, které tvoří základ navrhovaného softwaru.
- **Pohled procesů.** Modeluje procesově orientovanou variantu logického pohledu.
- **Pohled implementace**. Modeluje soubory a komponenty, které utvářejí hotový kód softwaru. Slouží jednak ke znázornění závislosti mezi jednotlivými komponentami, jednak toho, jak spravovat konfiguraci množin vytvořených z těchto komponent. Dále umožňuje definovat verzi softwaru.
- **Pohled nasazení**. Modeluje fyzické nasazení komponent na množinu výpočetních uzlů (počítačů a periferních zařízení).
- **Pohled případů užití**. Všechny ostatní pohledy jsou odvozeny od pohledu případů užití. Tento pohled zachycuje požadavky kladené na příslušný software a utváří tak základ tvorby dalších pohledů.

Po úvodu základních pojmů jazyka UML se nyní zaměříme na další nástroje, které jsou v práci použity. Jakou jsou modely případů užití, diagramy tříd, diagramy nasazení a sekvenční diagramy.

Modelování případů užití

Modelování případů užití je jednou z forem získávání požadavků. Umožnuje doplňkovým způsobem získávat a dokumentovat systémové požadavky. Modelování případů užití se skládá z následujících aktivit:

- Nalezení hranic systému, což je ohraničení kolem případů užití, jež je vyznačením území modelovaného systému.
- Vyhledání aktérů, což jsou role přidělených osobám a předmětům používající daný software.

- Nalezení případů užití. Což jsou činnosti, které mohou aktéři se softwarem vykonávat.
- Vytvoření relací. Smysluplné vztahy mezi aktéry a případy užití. (Arlow & Neustadt, 2007)

Modely případu užití poskytují hlavní zdroj objektů a tříd. Jsou prvotním vstupem k modelování tříd.

Diagram tříd

Diagram tříd zachycuje všechny třídy, které se týkají modelovaného systému a vztahy mezi nimi. (Buchalcevová, et al., 2007)

Diagram nasazení

Zobrazuje rozložení jednotlivých softwarových komponent na hardwarových zdrojích a jejich spolupráci. Rozmístění hardwarových a softwarových prostředků v lokalitách, topologie používaných sítí, druhy a využití komunikačních prostředků. (Arlow & Neustadt, 2007)

Sekvenční diagramy

Sekvenční diagram je nejpoužívanější UML diagram pro dynamické modelování. Znázorňuje grafický průběh zpracování v systému v podobě zasílání zpráv. Zprávy si posílají většinou objekty, ale mohou být posílány i třídě nebo aktérům. Základní charakteristikou objektu je schopnost přijmout zprávu a reagovat na ní. To značí, že v objektu probíhají určité operace, v rámci které objekt může poslat zprávu dalšímu objektu. Tak vzniká sekvence zpráv, kterou zachycuje právě sekvenční diagram. (Buchalcevová, et al., 2007)

Zprávy můžeme dělit na synchronní a asynchronní. Synchronní zpráva je taková, kdy vysílající objekt čeká na ukončení zpracování zprávy. Pokud objekt vyšle zprávu a nečeká na ukončení jejího zpracování, potom se jedná o asynchronní zprávu. (Buchalcevová, et al., 2007)

Diagram komunikace

Zachycuje instance tříd, jejich vzájemné vztahy a tok zpráv, které mezi nimi probíhají. Jsou podobné sekvenčním diagramů, ale zachycují jiné strukturální aspekty modelovaného systému. (Arlow & Neustadt, 2007)

Nyní zde jsou uvedeny základní pojmy, které v kontextu této práce pokládáme za důležité. Jednak proto, že se týkají dané problematiky, a jednak z důvodu pochopení významu pojmů, které byly použity v této práci. Dále jsou popsány jednotlivé programovací jazyky, které pro softwarové inženýrství slouží jako nástroj při navrhování a implementaci řešení.

2.3 Základní pojmy

2.3.1 Internet

Vlastní slovo Internet je zkratkou slova internetwork (asi bychom mohli říci „prostředí propojených počítačových sítí“, nebo „síť počítačových sítí“). Původním záměrem bylo vytvořit univerzální systém pro agenturu Advanced Research Project Administration ministerstva obrany USA (DARPA), následovalo jeho rozšíření na akademickou půdu a od počátku 90. let se Internet stal neustále rostoucím informačním prostředím umožňujícím komunikaci mezi nesčetným množstvím počítačů na celém světě. Internet nemá majitele, je to sada různorodých počítačových sítí propojených prostřednictvím bran, které zpracovávají přenos dat a převod zpráv a komunikují spolu prostřednictvím sady protokolů TCP/IP. (Kristián, 2001)

2.3.2 Webová aplikace

Na internetu, v literatuře a v dalších zdrojích můžeme nalézt velké množství definic webové aplikace. My zde uvádíme tuhle dle (managementmania.com, 2015) :

„Webová aplikace je taková aplikace, kterou není nutno instalovat na zařízení uživatele (počítač, tablet, smartphone) a můžete ji spustit z kteréhokoliv zařízení pomocí webového prohlížeče, protože je spuštěna na straně serveru. Vzhledem k tomu, že je třeba jen prohlížeč, se webová aplikace někdy nazývá též jako tenký klient.“

Právě rozmach internetu, internetových prohlížečů, rychlého připojení, nových technologií a programovacích jazyků umožnilo rozšíření webových aplikací.

V praxi se webová aplikace může na první pohled jevit jako jednoduchá statická webová stránka, která prezentuje pouze statické informace. V některých případech se jedná o složitější aplikaci provádějící komplikovanější úlohy za pomocí databázových systémů. Webové aplikace dále mohou být napojeny na další aplikace v organizaci (např. různé ekonomický softwary, či podpůrné systémy pro řízení organizací atd.). V dnešní době moderní webová aplikace umí téměř to, co software nainstalovaný na počítači. Nejpopulárnější příklady

webových aplikací jsou Facebook.com, LinkedIn.com, mezi poštovními programy (Gmail, Yahoo), kancelářské programy (Google Docs, Office365), různé intranety a celá řada dalších. Určit přesnou hranici mezi tím, co je webová stránka a co už je webová aplikace je někdy nemožné. (managementmania.com, 2015)

- Výhody webových aplikací:
  - Instalace není nutná,
  - uživatelé nemusí aktualizovat (aktualizace probíhá na serveru),
  - zapotřebí je pouze webový prohlížeč,
  - server zajišťuje uchovávání a zálohování dat, která jsou přístupná odkudkoliv.
- Nevýhody webových aplikací:
  - Nutnost připojení na internet,
  - rychlost aplikace je závislá na kvalitě připojení, při nekvalitním připojení může být práce s aplikací pomalejší,
  - v případě nekvalitního poskytovatele jsou možná bezpečnostní rizika v oblasti úniku dat. (managementmania.com, 2015)

Moderní webové aplikace prosazují tzv. dynamický obsah, který je označován jako Web 2.0, web druhé generace. Tyto aplikace mají označení Rich Internet Applications (RIA). Rozvoj směrem k RIA odstartovala společnost Google v roce 2005, když představila služby Google Mail a Google Maps. Tyto webové aplikace se svým ovládáním a funkcemi podobají stolním offline aplikacím. Těchto funkcí dosáhl Google pomocí nasazení technologie Ajax. Ajax (Asychnorouns JavaScript and XML) je několik technologií pro vývoj webových aplikací, jejíž významnou odlišností od statických webových stránek je, že není potřeba vždy obnovovat celou stránku, nýbrž jen předem určené specifické části. Podle (Gerrat,2015) Ajax přestavuje spojení těchto technologických částí:

- HTML a CSS (standardní technologie pro prezentaci webových stránek),
- výměna a manipulace dat pomocí XML,
- asynchronní načítání dat pomocí XMLHttpRequest (XHR),
- JavaScript, který spojuje všechny technologie dohromady.

Pro úplnost, zde zmiňujeme základní popis uvedených technologií. Jak uvádí (Basham, Sierra, & Bates, 2008), HTML (HypertText Markup Language) je hlavním prostředkem pro

vytváření webových stránek, popisuje, jak má webová stránka vypadat a jaké má mít funkce. Dále umožnuje publikaci dokumentů na Internetu. Je charakterizován množinou značek (tzv. tagů) a jejich atributů.

XML je značkovací jazyk určený pro výměnu dat mezi aplikacemi nebo pro uchování dokumentů. Modul XMLHttpRequest umožňuje výměnu dat nejen ve formátu XML mezi klientem a serverem.

XMLHttpReqeust zajišťuje výměnu dat na pozadí a JavaScript překreslí pouze části stránky, u které je to nutné. Touto základní vlastností Ajaxu můžeme využít tak, že se aplikace skutečné chovají jako destopové aplikace.

JavaScript je interpretovaný programovací jazyk pro webové stránky. Jsou jím obvykle ovládány různé interaktivní prvky (tlačítka, textové pole) nebo tvořeny animace a efekty obrázků.

Nyní se zaměříme na hlavní technologie, které byly při vývoji webové aplikace využity.

2.4 Technologie

V úvodu této kapitoly jsou popsány klíčové technologie, které byly při vývoji aplikované a použité. Pro realizaci aplikace byla zvolená kombinace Java technologií:

- Jasper Reports,
- Google web toolkit,
- Sencha GXT,
- Spring Framework,
- Hibernate Framework.

2.4.1 JasperReports

Jedním z cílů této práce je generování tiskových výstupů, které probíhají na základě tiskové sestavy. Pro realizaci této funkcionality byl zvolen JasperReports, jako jeden z dostupných nástrojů pro tvorbu výstupů na platformě Java. (Danciu & Chirita, 2007)

JasperReports (dále jen JR) je výkonný a flexibilní nástroj pro generování reportů, který poskytuje bohatý obsah elementů na obrazovce, tiskárně nebo do souboru ve formátu PDF, HTML, RTF, XLS, ODT, CSV, nebo XML. Knihovna je kompletně implementovaná v Javě a může být použita v různých Java aplikacích, včetně Java EE nebo webové aplikace pro

generování dynamického obsahu. Jejím hlavním účelem je vytvoření orientované stránky připravené pro tisk. (Danciu & Chirita, 2007)

JR, jako většina reportovacích nástrojů, používá šablony sestav strukturované ve více sekcích:

- titulek – určený pro nadpis reportů, který je uveden na první straně,
- hlavička stránky – záhlaví stránky,
- hlavička tabulky – záhlaví tabulky,
- hlavička skupiny,
- detail – určen pro jeden datový řádek, který se opakuje podle počtu záznamů získaných ze zdroje dat,
- patička skupiny,
- patička tabulky,
- shrnutí – závěrečné shrnutí celého reportu,
- pozadí – společné pro každou stránku.

Každý část má své vlastní uspořádání, ve kterém můžeme umístit různé typy elementů, jako jsou obrázky, statické a dynamické textové pole, čáry a obdélníky. Mimo těchto základních elementů jsou v nabídce i pokročilé elementy jako vložení HTML, Google mapy nebo křížové tabulky. (Danciu & Chirita, 2007)

Sestava se generuje na základě XML (JRXML) souboru, který představuje šablonu pro danou sestavu, definuje tedy vzhled a umístění dat. Šablonu můžeme zkompilovat do binární verze šablony (.jasper). Takto zkompilovaná šablona je serializovatelná a může být uložena na disk. To zamezuje modifikování výsledné sestavy a urychluje zpracování dat. (Danciu & Chirita, 2007)

Data pro tisk mohou pocházet z různých datových zdrojů, včetně relačních databází, kolekce, nebo pole objektů jazyka Java nebo dat z XML. Nově je k dispozici rozhraní pro Hibernate (více v kapitole Hibernate) a CSV. Získaná data je možné dále zpracovávat za pomocí uživatelských parametrů a proměnných. (Danciu & Chirita, 2007)

JR podporuje velké množství výstupních formátů. Podporovány jsou formáty PDF, HTML, XHTML, ODS, DOC, DOCX, ODT, RTF, PPTX, XLS, XLSX, XML, TXT, CSV. (Danciu & Chirita, 2007)

Pro JR existuje několik návrhářů, kteří slouží pro zjednodušení tvorby tiskových sestav. V našem případě byl použit „Jaspersoft Studio“, který umožňuje vizuální editaci XML šablon.

JR poskytuje mnoho funkcí v oblastech výběru, zpracování a prezentace dat. Je tedy výborným pomocníkem v návrhu tiskových výstupů. JR je dostupný pod licencí **LGPL**, takže je možné jej bez obav použít. Společnost Jaspersoft nabízí k tomuto produktu také placenou podporu, literaturu a semináře. Tento nástroj se bude s vysokou pravděpodobností dále rozvíjet, z čehož může aplikace v budoucnu čerpat další výhody.

Další důležitou technologií je Spring Framewok, který zprostředkovává spojení mezi Hibernate a databázovým úložištěm.

2.4.2 Spring Framework

Spring Framework je produktem organizace Spring Source kontrolovanou společností VMware. Cílem Spring Frameworku je zjednodušení a zefektivnění vývoje. Vývojáři se nemusí zaobírat infrastrukturou a mohou se přímo soustředit na aplikaci a její business logiku. Spring Framework poskytuje robustní infrastrukturní podporu. Jeho význačnou vlastností je to, že umožňuje budovat podnikové aplikace z tzv. POJO objektu (Plain Old Java Object). POJO je označení pro standardní objekty jazyka Java. Spring Framework pak umožňuje na tyto POJO objekty neinvazivně aplikovat podnikové služby. Výhodou Spring Frameworku je nízká míra jeho intruzivity, tedy skutečnost, že prostředky Frameworku nepronikají přímo do business logiky aplikace. (Walls, 2013)

Klíčové myšlenky Spring Frameworku jsou založené na návrhových vzorech – Inversion of Control a Dependency Injection.

Inversion of Control

Inversion of Control (IoC), nebo také obrácené řízení, je návrhový vzor, který umožňuje uvolnit vztahy mezi jinak těsně svázanými komponentami. V klasickém modelu programování vytváříme nějakou třídu, která pak využívá další třídy, a tak dále. Tím jsou potom třídy velmi pevně svázány a změna používané třídy za jinou vyžaduje zásah do kódu. IoC umožňuje uvolnit tuto vazbu, což nejjednodušeji vystihuje Hollywoodský princip „Nevolejte nám, my se ozveme.“ To je klasická odpověď, kterou dostávají amatérští herci, kteří se snaží získat v Hollywoodu nějakou roli ve filmu, seriálu apod. Proto tedy Hollywoodský princip. V podstatě to znamená, že třída nevytváří sama instance dalších tříd, které potřebuje, ale jsou jí dodány nějakým způsobem z vnějšku. (Walls, 2013)

Dependency Injection

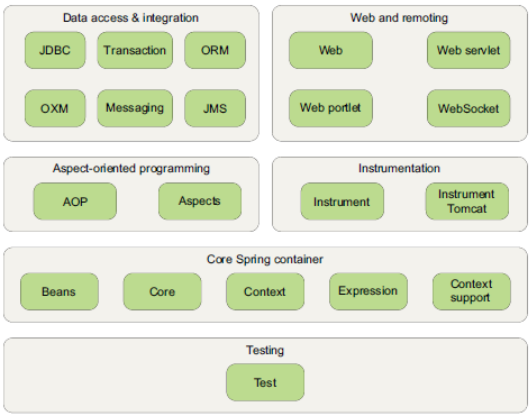
Dependency injection (DI), nebo také vkládání závislostí, je v podstatě novější název pro IoC, zužuje ale také okruh, na který se vztahuje. Jedná se už o konkrétní techniku využití IoC. DI má několik zásadních výhod pro programování, jako jsou zejména tyto:

- Jednodušší psaní a údržba jednotlivých komponent, díky vzájemné menší závislosti.
- Je výrazně jednodušší kód testovat. Není třeba starat se o přetypování tříd, protože se o něj stará IoC kontejner. Závislosti komponent jsou definovány explicitně, je tedy jednodušší se v nich vyznat.
- IoC kontejner ve většině případů nevyžaduje speciální zásah do kódu aplikace, jednotlivé komponenty je pak možné snadno využít i jinde bez nutnosti přepisování kódu. (Walls, 2013)

Celý Spring Framework je sestaven z několika modulů, které jsou organizovány do šesti skupin:

1. Core Spring Container
2. Data Acces/Integration
3. Web and remoting
4. AOP (Aspect-oriented Programming)
5. Instrumentation
6. Testing

Hlavní výhodou Frameworku je modularita, která poskytuje vývojáři využít jen ty skupiny modulů, které zrovna potřebuje. Kategorii Spring Frameworku ilustruje níže uvedený obrázek č. 6. Spring Framework poskytuje celou řádu užitečných nástrojů a služeb – deklarativní transakce, integraci s Hibernate (popsán níže), vlastní MVC (Model – view – controller) Framework. (Walls, 2013)



Obrázek 6: Kategorie modulů Spring Frameworku (Zdroj: Walls, 2013, s 22)

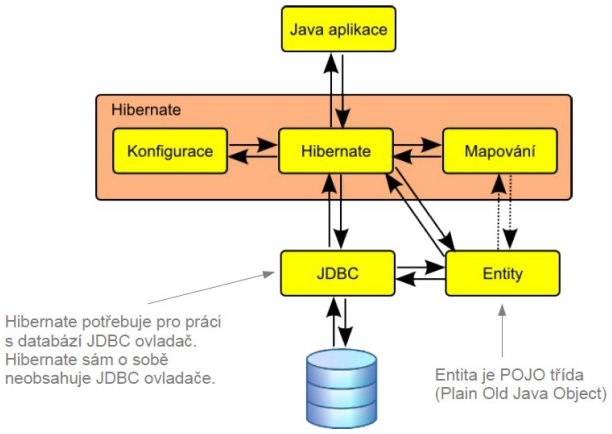
2.4.3 Hibernate

Hibernate Framework je jedním z nástrojů pro objektově-relační mapování (ORM). Vznikl v roce 2001 a v současnosti je podporován společností Red Hat. Jeden z hlavních cílů Hibernetu je perzistence dat, tedy ukládání a uchování dat, aby byly k dispozici po ukončení běhu programu. (Bauer & King, 2015)

Hibernate implementuje standard JPA (Java Persistance API) a přidává k němu další možnosti. Způsob objektově-relační mapování umožňuje mapovat objekty používané v aplikaci na tabulky v relační databázi. Díky tomu vývojář pracuje s daty v databázi jako s objekty a v ideálním případě se nemusí příliš starat o jejich životní cyklus. To jeho práci v podstatě zefektivňuje a zpříjemňuje. Mapovat lze veškeré tyty vazeb, které se objevují v relačních databázích (1:1, 1:N a M:N). Mapování můžeme nastavit prostřednictvím tzv. anotací zapsaných přímo v kódu mapované třídy nebo pomocí konfiguračního souboru. Hibernate dále nabízí bohaté možnosti dotazování. K datům v databázi můžeme přistupovat pomoci objektového dotazovacího jazyka HQL (Hibernate Query Language), pomocí kritérií QBC (Query By Criteria), nebo prostřednictvím nativního SQL. (Bauer & King, 2015)

Hiberate autonomně řídí způsob přístupu k databázi, které je založeno na interním využití JDBC. Architektura Hibernate používá dvě specializovaná rozhraní – Session pro jednorázové připojení k databázi a Transaction, které abstrahuje JDBC transakce. Tato dvě rozhraní spolu s rozhraním pro dotazování umožňuje komunikaci s aplikací a s databází. (Bauer & King, 2015)

Pro lepší znázornění můžeme na obrázku 7 vidět role Hibernate v Java aplikaci.



Obrázek 7: Role Hibernate v Java aplikaci (Zdroj: Bauer & King, 2015)

Vhodná implementace Hibernate nám poskytne nezávislost na databázi. Pokud je vše správně implementováno, bude v případě přechodu na jinou databázi postačující vykonat dva

kroky: provést export datového schématu pro nově zvolenou databázi nebo v konfiguračním souboru nastavit export datového schématu a v konfiguraci Hibernate změnit driver databáze. Více není potřeba. Pokud je vše správně implementováno, aplikace bude fungovat zcela stejně jako na vývojové databázi, přičemž náklady na instalaci budou shodné s běžnými náklady, tj. nebudou zde žádné nadbytečné práce spojené s přechodem na jinou platformu. Výjimku tvoří databázové procesy určené pro rozsáhlé datové operace, které se vždy při první implementaci budou muset přizpůsobit dané databázové platformě.

Nyní se zaměříme na zvolení správné webové technologie, která uspokojí všechny požadavky a funkce uživatelského rozhraní. Bylo zapotřebí vybrat framework s významnou podporou AJAX a mnoha pokročilými funkcemi a komponentami. Těmto požadavkům dobře vyhovuje Google Web Tollkit (GWT).

2.4.4 Google Web Toolkit

GWT je moderní nástroj pro tvorbu dynamických webových aplikací na platformě Java. Hlavní myšlenkou tohoto frameworku je umožnit vývojářům vytvářet moderní a rychlé aplikace, aniž by museli znát specifikace prohlížečů a skriptovací jazyk JavaScript. Toto je zabezpečeno tím, že GWT v podstatě překládá jazyk Java do JavaScriptu, který je posléze spuštěn na klientském prohlížeči a zajišťuje komunikaci se serverovou částí aplikace. Grafiku lze vytvářet na serveru a klientovi posílat hotový výstup. Části aplikace vyžadující více spolupracujících vláken mohou být vykonávány na serveru. V klientské části kompilované do JavaScriptu je nutné mít na paměti odlišnosti, které jsou ale velmi dobře zdokumentovány. (Tacy, et al., 2013)

Maximální důraz klade GWT na optimalizaci přenášených dat mezi klientem a serverem. Pro každý podporovaný prohlížeč se kompilují odlišné verze kódu. Pro uživatelské rozhraní používá vlastní knihovny, které jsou uzpůsobeny webovému prostředí.

GWT disponuje dobrým zázemím, které zahrnuje vynikající dokumentaci, dostupné nástroje pro vizuální návrh a podporu všech hlavních vývojových nástrojů pro Javu. K dispozici jsou i nadstavby třetích stran, které přidávají novou funkcionalitu nebo vylepšují stávající. Tento systém přináší pro vývojáře mnoho výhod, protože lze využívat ladící nástroje pro platformu Java. GWT je tvořeno čtyřmi klíčovými komponentami:

- GWT kompilátor Javy do JavaScriptu,
- hostovaný webový prohlížeč,
- emulační knihovna JRE,
- knihovna prvků uživatelského rozhraní (UI).

GWT kompilátor zabezpečí při kompilaci překlad jazyka Java do optimalizovaného kódu v jazyku JavaScript. Výhody tvorby aplikací v GWT jsou následující:

1. Tvorba aplikací je podobná klasickému vývoji desktopových aplikací v jazyku Java, která usnadní práci programátorskému týmu, který využívá tento jazyk.

2. Kompilace zabezpečuje kompatibilitu různých variant JavaScriptu.

3. Vykreslovaní stránky probíhá na straně klienta, server není zbytečně zatěžován a je schopen obsloužit více požadavků.

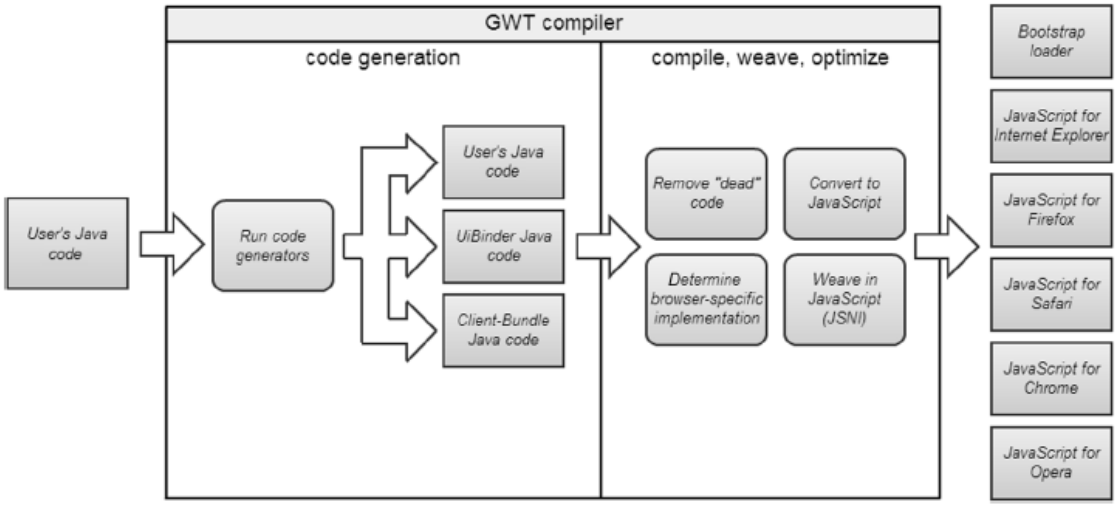
Nevýhody:

1. Obsah je generovaný na straně klienta pomocí JavaScriptu, což vede k problému optimalizování stránky pro vyhledávací nástroje (Google, Yahoo Search apod.). Tento problém můžeme řešit tvorbou alternativních statických stránek pro vyhledávače, které budou obsahovat potřebné informace.

2. Velká zátěž klienta.

3. Výsledkem je poměrně velké množství kódu napsaného v jazyku Java.

Proces GWT kompilátoru lze vidět na obrázku číslo 8.



Obrázek 8: Proces GWT kompilátoru (Zdroj: Tacy, Hanson, Essington, & Tökke, 2013)

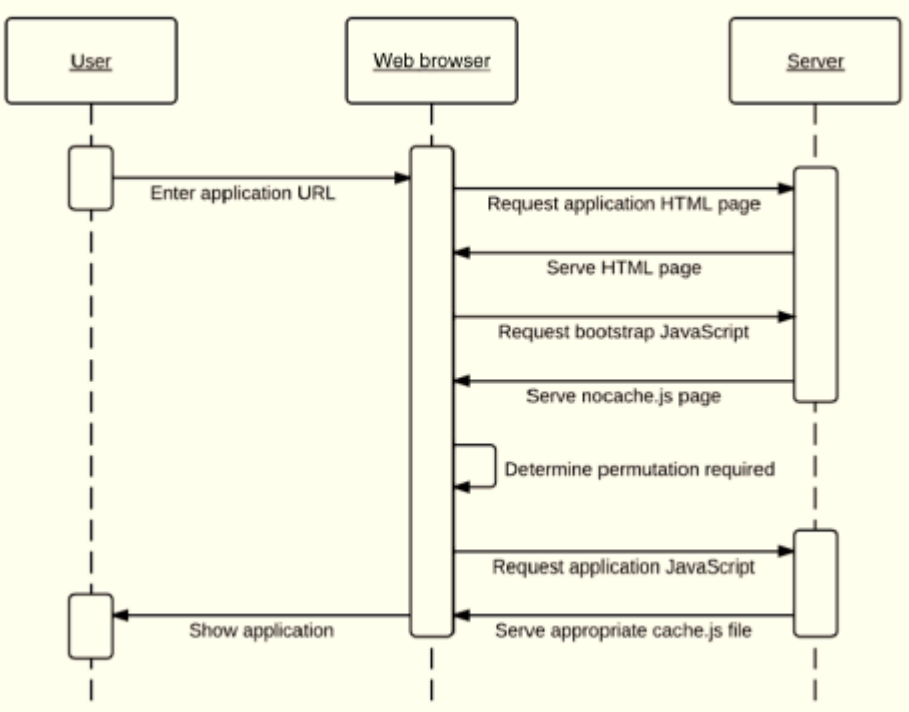
Jak lze vidět na obrázku č. 8, kompilátor vygeneruje více výstupních souborů JavaScript, protože JavaScript se chová v každém prohlížeči trochu jinak.

Hostovaný webový prohlížeč umožňuje spouštět vyvíjenou aplikaci lokálně v JVM (Java Virtual Machine) ještě před jejím přeložením do JavaScriptu. Takto spuštěná aplikace

běží jako Java kód, díky tomu můžeme využívat jakékoliv nástroje obsažené v našem vývojovém prostředí. Emulační knihovna obsahuje implementace některých tříd platformy Java v JavaScriptu. Knihovny Java, které nejsou zahrnuté v emulační knihovně, nelze používat. Uživatelské rozhraní obsahuje třídy přestavující tzv. widgety – jednotlivé komponenty uživatelského rozhraní, jako jsou panely, tabulky, popisky, tlačítka. (Tacy, et al., 2013)

Životní cyklus GWT aplikace

Uživatel pomocí URL adresy webového prohlížeče požádá o spuštění aplikace (obvykle HTML stránka). Na obrázku 9 můžeme vidět celý proces spuštění aplikace.

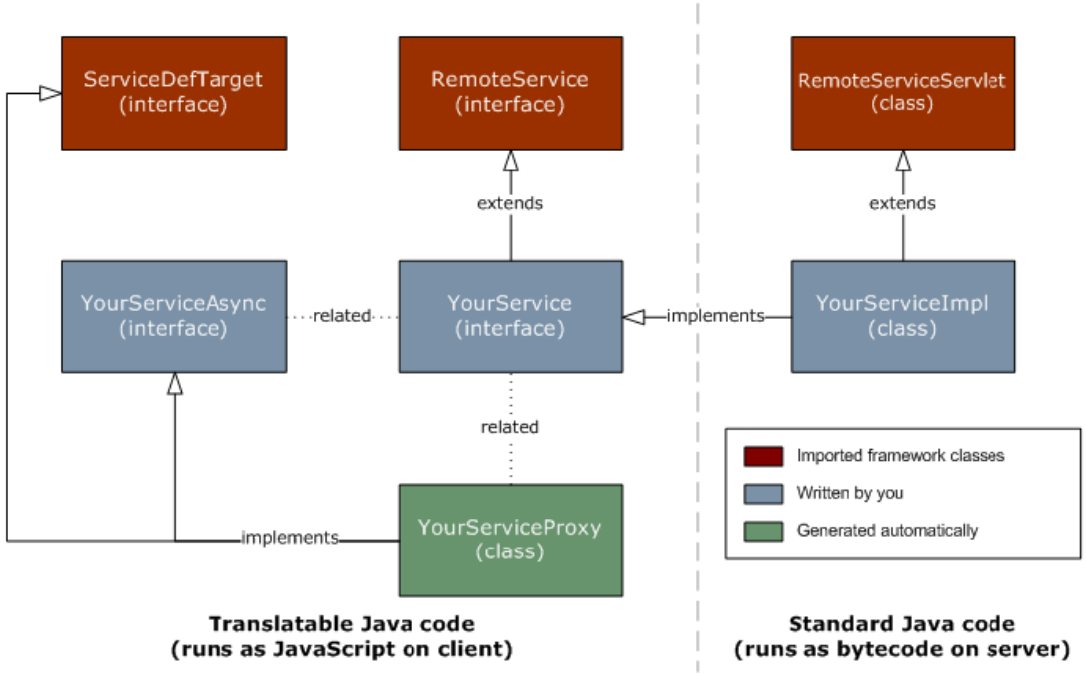


Obrázek 9: Sekvenční digram ukazující požadavek uživatele na spuštění GWT aplikace (Zdroj: Tacy, Hanson, Essington, & Tökke, 2013)

HTTML stránka obsahuje JavaScriptový soubor nochache.js file – tzv. bootstrap soubor. Bootstrap kód určuje konkrétní permutaci aplikace v JavaScriptu. Po načtení bootstrap volá kompilovanou metodu onModuleLoad, které je vstupním bodem aplikace. (Tacy, et al., 2013)

Komunikace se Serverem

Neoddělitelnou částí GWT aplikace je komunikace se serverem. Bez této komunikace nemůžeme uskutečnit například generaci tiskových výstupů na straně serveru nebo perzistenci dat. GWT má zabudovaný protokol GWT RPC (GWT remote procedure call), který slouží pro volání definovaných metod na serveru a zabezpečuje tak asynchronní komunikaci. Na obrázku lze vidět postup implementace GWT RPC do aplikace. (Tacy, et al., 2013)



Obrázek 10: Implementace RPC GWT do aplikace (Zdroj: Tacy, Hanson, Essington, & Tökke, 2013)

Obrázek č. 10 znázorňuje 2 logicky oddělené části, a to klientskou (zkompilovaný JavaScript) a serverovou, které běží jako Java bytecode na serveru.

GWT je skvělý způsob jak vytvořit webovou aplikaci založenou na technologii AJAX, bez nutnosti znalosti JavaScriptu a zároveň se vypořádat s nekompatibilitou prohlížečů. Nicméně je to nástroj, který tvoří součást řešení spíše než celkové řešení.

Proto pro samotné komponenty uživatelského rozhraní je výhodné vybrat některou z dostupných sad komponent. Tyto komponenty mají mnohem větší možnosti než ty, které jsou standardní součástí čistého GWT. Zejména je třeba se zaměřit na dostupnost pokročilých funkcí bohatého uživatelského rozhraní, které budeme potřebovat při realizaci našeho návrháře. Pravděpodobně nejvyspělejšími komponentami v této technologii jsou komponenty

od společnosti Sencha. Dříve byly známé pod názvem Ext GWT (přitom ”GWT Ext“ je jiný produkt, který je mnohými začátečníky v této oblasti často zaměňován) a nyní jsou vydávány pod názvem Sencha GXT.

2.4.5 Sencha GXT

Sencha GXT je nejkomplexnější Java Framework pro vytváření funkčně bohatých webových aplikací. Používá GWT překladač, který umožňuje vývojářům psát aplikace v jazyce Java a sestavit svůj kód do vysoce optimalizovaného HTML5 kódu, který je schopen zobrazení obsahu napříč různými prohlížeči, zařízení a velikosti obrazovky. (Sencha.com, 2015)

Sencha GXT je vybaven vysoce výkonnými UI komponentami, které jsou interoperabilní s nativní součástí GWT. Tyto komponenty jsou zcela přizpůsobitelné a patří zde například panely nástrojů, vyskakovací okna, formuláře a mnoho dalších. Pokud nenajdeme komponentu, kterou hledáme, stovky rozšíření jsou k dispozici na Sencha komunity. (Sencha.com, 2015)

Dále pokročilý balíček grafů umožňuje vizualizovat velké množství dat s širokým množstvím a s širokým spektrem typů grafů. Tyto grafy se vykreslují pomocí SVG, VML. Varianty prohlížeče jsou zpracovány automaticky, takže grafy se budou vždy zobrazovat správně. (Sencha.com, 2015)

Framework obsahuje vestavěnou podporu pro RPC a JSON, takže lze přistupovat do aplikací pomocí libovolného zdroje dat. Data mohou být na straně klienta ukládány do vysoce funkčních modelů, které nabízejí funkce jako je třídění a filtrování. Sencha GXT rovněž nabízí kompletní podporu motivů, což nám umožňuje vytvářet a graficky upravovat webové aplikace. Motivy používají jednoduchý konfigurační systém zpřístupňující stovky proměnných, které mohou být změněny. (Sencha.com, 2015)

Nicméně pokud využijeme tuto kombinaci webových technologií, nebude nutné instalovat žádné dodatečné doplňky do prohlížeče. Tyto technologie nám umožní realizovat i pokročilé funkce jako přesné nastavování pozic a velikostí jednotlivých elementů formou WYSIWYG stejně jako online úpravy v datových tabulkách pro rychlou úpravu vlastností vybraných objektů.

2.5 Nástroje pro implementaci

Pro jazyk Java existuje mnoho kvalitních vývojových prostředí. Nejznámější vývojová prostředí jsou NetBeans a Eclipse (v obou případech open-source aplikace). Pro realizaci bylo zvoleno Eclipse vývojové prostředí z důvodu autorových zkušeností. Mimo jiné Eclipse je podporován společností Google, které je vlastníkem technologie Google web toolkit. Pomocí plaginů do Eclipse zabezpečíme podporu standardizovaných Maven projektů. Mnoho reálných aplikací je vyvíjeno formou Maven projektu a pokud má být integrace veškerých technologií co nejsnadnější, je velmi užitečné mít tento projekt také v Maven formě.

2.5.1 Definice projektů pomocí Maven

Maven představuje modul od společnosti Apache. Klade si za cíl sjednotit a zjednodušit správu projektů. Stará se o natavení, kompilaci a získávání potřebných informací pro vedení projektu. Základem pro budování projektu je soubor pom.xml (Project Object Model), který obsahuje veškeré informace o zdrojových souborech projektu (struktura, verze, závislost), potřebné knihovny a další návaznosti na vývojové prostředí. Díky tomu se Maven postará sám o sestavení distribuce aplikaci. Programátor si může vybrat z několika možností (maven-plugin, ejb, war, ear, rar, a par). Maven si stahuje potřebné soubory z Internetu, proto potřebuje pro první spuštění přístup k aplikaci, aby mohl stáhnout potřebné závislosti a nástroje pro sestavení projektu a následně projekt správným způsobem sestaví. (Varanasi & Belida, 2014)

3 Analýza business požadavků

Dříve než zahájíme analýzu, návrh a kódování, je potřeba identifikovat požadavky, které má webová aplikace splňovat. Musíme zjistit, jaké funkce bude uživatel s programem realizovat, za jakých podmínek, v jakém prostředí.

Průzkumy zaměřené na vývoj softwaru ukazují, že více než 30 % všech softwarových projektů je zrušeno před dokončením a přes 70% projektů nedodá požadovanou funkcionalitu. (Buchalcevová, et al., 2007)

Základem těchto problémů je správa požadavků. Pokud nejsem schopni pochopit navrhovaný software, pak nemůžeme specifikovat jeho chování, nedokážeme ho navrhnout, implementovat, testovat a taky definovat očekávané přínosy. (Buchalcevová, et al., 2007)

Chyby, které jsou způsobené při shromažďování požadavků, jsou propagovány do všech následujících fázích vývoje (*vysvětluje to dříve zmíněna metodika vývoje vodopád*). Vývoj pak probíhá technicky správně, ovšem podle špatného zadání. Špatné zadání pak přebírají pracovníci dalších profesí a tráví čas na něčem, co nebylo požadováno nebo alespoň mělo fungovat jinak. Pokud se chyba neodhalí ani v době testování, pak se zbytečně promarní další peníze a drahocenný čas. (Wiegers, 2008)

Vzhledem k tomu, že je konečná webová aplikace založena na množině požadavků, je efektivní inženýrství požadavků klíčovým faktorem celého vývoje projektu.

Nejprve je nutné vymezit pojem požadavek. Wigers nabízí hned několik alternativních definic od různých autorů. Definice podle softwarové terminologie (IEEE Standard Glossary of Software Engineering Terminology) z roku 1990 definuje požadavek jako:

1. *Podmínku nebo funkci, kterou uživatel potřebuje pro řešení problému nebo dosažení nějakého cíle.*
2. *Podmínku nebo funkci, kterou musí systém nebo jeho část splňovat, aby vyhověl smlouvě, standardu, specifikaci nebo jinému dokumentu, jenž se na něj formálně vztahuje.*
3. *Dokumentovanou podobu některého z předchozích dvou bodů.“*

Tato definice zahrnuje uživatelův i vývojářův pohled na požadavky. Další definice podle (Sommerville a Sawyer,1997):

„Požadavky jsou (...) popis toho, co všechno by se mělo implementovat. Popisují žádané chování systému a jeho vlastnosti a mohou představovat nějaká omezení procesu vývoje systému.“

Je zjevné, že žádná univerzální definice požadavků není. V této práci se přikláníme na definici: *„požadavek je specifikací toho, co by mělo být implementováno“*. (Arlow & Neustadt, 2007)

3.1 Specifikace softwarových požadavků

Specifikace softwarových požadavků (SSP) je úplným začátkem procesu tvorby softwaru. Obecně se považuje za počáteční vstup k objektové analýze a následnému objektovému návrhu. SSP obsahuje *model požadavků,* který podle (Arlow & Neustadt, 2007) rozlišuje dva typy požadavků:

- funkční požadavky, jež určují, jaké chování bude software nabízet,
- nefunkční požadavky, které specifikují vlastnosti nebo omezující podmínky daného softwaru.

Dále SSP obsahuje *model případu užití,* ve kterém nalezneme mnoho balíčků s případy užití. Jejím obsahem jsou případy užití, neboli specifikace funkce softwaru, aktéři (externí role s přímou interakcí se softwarem) a relace. (Arlow & Neustadt, 2007)

3.2 Atributy požadavků

Každý požadavek muže obsahovat určitou množinu atributů, které zachycují dodatečné informace týkající se příslušeného požadavku. Snad nejznámějším atributem požadavků je *priorita.* Hodnota tohoto atributu je relativní priorita vůči ostatním požadavkům. Obvyklé schéma přiřazování priorit nalezneme v tabulce číslo 3.

Priorita	**Sémantika**
Nezbytný (Must have)	Povinné požadavky, jenž tvoří základ softwaru.
Možný (Should have)	Důležité požadavky, avšak možné vynechat.
Eventuální (Could have)	Požadavky, které jsou nepovinné (provádí se, je-li čas).
Chceme mít (Want to have)	Požadavky, jež jsou zahrnutý do dalších verzí softwaru.

Tabulka 3: Hodnoty atributu priorita (Zdroj: Arlow & Neustadt, 2007)

Při použití uvedeného schématu, může mít atribut, *priorita* jednu z následujících hodnot: M, S, C nebo W. Mnoho nástrojů pro správu požadavků umožňují formulovat dotazy na základě hodnoty atributů. Díky tomu může generovat seznam nejdůležitějších požadavků. (Arlow & Neustadt, 2007)

Přesné stanovení sady požadavků závisí na povaze a potřebách konkrétního projektu. Pro definici množiny atributů je ale důležité, aby atributy byly co nejjednodušší. Volíme pouze ty atributy, které budou pro náš projekt prospěšné. Snažíme se počet atributů udržovat na minimu. (Arlow & Neustadt, 2007)

3.3 Shrnutí a analýza požadavků společnosti u&sluno

V této kapitole jsou popsány jednotlivé požadavky společnosti U&SLUNO na webovou aplikaci, které jsou rozděleny podle funkčnosti a priorit. V tabulce číslo 2 nalezneme rozdělení a popis veškerých požadavků. Atributy pro popis požadavků byly vybrány identifikační číslo požadavku (ID), název, priorita a kategorie. Po rozdělení jednotlivých požadavků, je vytvořena tabulka číslo 3 s konkrétními případy užiti, v tabulce číslo 4 nalezneme RTM matici.

Nejprve je potřeba upřesnit požadavky, které bude navržený modul zahrnovat.

3.3.1 Popis a rozdělení požadavků

Nejprve se zaměříme na tvorbu konkrétních výstupů, která obvykle probíhá na základě existující tiskové šablony. Tisková šablona je přepis, který společně s konkrétními daty umožňuje vytvořit konkrétní tiskovou sestavu. V běžném informačním systému plní tiskové sestavy především funkce výběru dat, zpracování a její následné prezentaci. Tyto tři oblasti jsou zásadní a každý reportovací nástroj nabízí různou úroveň funkcí pro každou z nich.

ID	Název požadavku	Priorita	Kategorie
R1	Výběr dat	M	Funkční
R2	Zpracování dat	M	Funkční
R3	Grafické rozdělení reportu	M	Nefunkční
R4	Editace stránky	M	Funkční
R5	Grafické umísťování elementů	M	Funkční
R6	Ukládání tiskové sestavy do souboru	M	Funkční

R7	Načítaní tiskové sestavy ze souboru	M	Funkční
R8	Definice standardních elementů	M	Funkční
R9	Generování reportu	M	Funkční
R10	Náhled reportu v prohlížeči	M	Funkční
R11	Software bude ve formě webové aplikace	M	Nefunkční
R12	Přehlednost a intuitivní ovladatelnost	M	Nefunkční
R13	Stromová struktura obsahu reportu	M	Funkční
R14	Odezva	M	Nefunkční
R15	Logovací mechanizmus	M	Funkční
R16	Nastavení pozice a velikost jednotlivých elementů	M	Funkční
R17	Ochrana dat	W	Nefunkční
R18	Vizualizace dat pomocí grafů	W	Funkční
R19	Vkládání, načítání a ukládání obrázků	W	Funkční

Tabulka 4: Shrnutí požadavků na aplikaci (Zdroj: vlastní zpracování)

R1 – Výběr dat - funkce výběr dat obsahuje získání dat. Většinou se jedná o čtení dat z databáze přímého spojení (JDBC, Hibernate), nebo ze souboru (CSV, XML, TXT, JSON). Tato data je možné určit výběrovým dotazem (SQL, XPath), případně je dodatečně omezit různými podmínkami nebo je dále seskupovat dle daných kritérií. Častá funkce je vložení do šablony tzv. parametry, které mohou být vkládány automaticky nebo jako uživatelský vstup.

R2 – Zpracování dat - Funkce pro zpracování dat umožňují zejména použití agregačních funkcí (součty, celkové počty, průměry, minima, maxima, atd.). Někde je zapotřebí i pokročilé zpracování jednotlivých údajů pomocí tvorby vlastních funkcí, obzvlášť využitím jazyka Java (funkce pro práci s řetězci, čísly, apod.). Pokročilé nástroje umožní i průběžné vlastní výpočty během sestavování reportu, které jsou udržované v uživatelských proměnných. Jejich hodnoty je pak možné v tiskové sestavě využít stejně jako vybraná data.

R3 - **Grafické rozdělení reportu** - Tisková sestava se obvykle skládá ze sekcí, které se v závislosti na typu mohou opakovat. Sekce jsou vzájemně oddělené oblasti s definovanou velikostí (výškou) a logickým významem. Šířka je definována velikostí výstupního formátu. Na základě typu sekce může být sekce povinná (detail) nebo volitelná (patička stránky). Sekce se vypisují a opakují automaticky na základě zvolené velikosti výstupu a množství prezentovaných dat (např. na každé nové stránce je nejprve vytisknuta sekce „hlavička

stránky“). Obsahem jednotlivých sekcí jsou pak objekty sloužící pro vlastní prezentaci dat, dále označované jako Elementy. Základními sekcemi tiskové sestavy jsou:

- titulek (hlavička sestavy),
- hlavička stránky,
- hlavička tabulky,
- hlavička skupiny,
- detail (řádek výpisu dat),
- patička skupiny,
- patička tabulky,
- patička stránky,
- shrnutí (patička sestavy).

R4 – Editace stránky – Funkce editace stránky bude umožňovat výběr ze standardních formátů papíru, s možností editace okrajů stránky a orientace (landscape, portrait). Dále umožní rozdělit stránky do několika sloupců. Ve výchozím nastavení bude aplikace vytvářet reporty s jedním sloupcem na každé stránce. Podle orientace stránky a velikosti stránky vypočte rozmístění reportů na stránku.

R5 – Grafické umísťování elementů – Prostřednictvím funkce umísťování elementů bude možné vkládat do sekce elementy (objekty), které zajistí vlastní prezentaci dat. Prezentovaný obsah každého elementu může být předem daný (statický, jako součást šablon) nebo vkládaný do šablony automaticky až v době generování tiskové sestavy (dynamicky, vkládaný z vybraných dat). Základní elementy tiskové sestavy, které obsahují všechny plnohodnotné reportovací nástroje, jsou:

- textové pole,
- statický text,
- čára (vodorovná i svislá),
- obdélník,
- obrázek,
- čárový kód.

U datových elementů a standardních elementů bude možné určit styl písma, barvu, pozici a velikost písma (včetně čárového kódu).

R6 - Ukládání tiskové sestavy do souboru – Aplikace bude schopna ukládat grafický návrh tiskové sestavy do souboru (xml) na lokální úložiště nebo do databázového systému.

R7 - Načítaní tiskové sestavy ze souboru – Aplikace umožní načítaní uložených tiskových sestav do webové aplikace. Bude schopná přečíst šablonu a následně vytvořit veškeré elementy ve webové aplikaci a umožní jejich manipulaci.

R8 - Definice standardních elementů – V aplikaci bude možné vybrat standardní elementy tiskových sestav: číslo stránky, počet stránek, aktuální datum, aktuální čas.

R9 - Generování reportu – Aplikace bude schopna prezentovat data pomocí generovaného výstupu, tzv. výstupní formáty. Moderní generátory podporují mnoho výstupních formátů, do kterých je výsledek uložen. Aplikace bude umožňovat ovlivnit výběr výstupního formátu, přičemž některé formáty jsou vhodnější pro finální prezentaci, jiné pro další zpracování:

PDF, PPT, HTML, XHTML, RTF, DOC, DOCX, ODT (formáty vhodné pro prezentaci),

XML, XLS, XLSX, CSV, TXT (formáty vhodné pro zpracování).

R10 - Náhled reportu v prohlížeči – Funkce náhled reportu bude umožňovat zobrazení vygenerovaného reportu ve formátu PDF ve webovém prohlížeči.

R11 - Software bude ve formě webové aplikace – Jeden z hlavních cílů této diplomové práce.

R12 - Přehlednost a intuitivní ovladatelnost – Aplikace bude přehledná a intuitivní i pro méně zkušené uživatele. Uživatelské rozhraní musí umožňovat snadnou a rychlou tvorbu nových šablon tiskových sestav a jejich následné úpravy. Dále musí splňovat vlastnosti WYSIWYG, tedy „co vidíš, to dostaneš“. Uživatel očekává, že během přípravy šablony uvidí, jak bude vypadat výsledná tisková sestava.

Funkce uživatelského rozhraní musí zajistit dostatečný komfort při vytváření a úpravách tiskových sestav. Požadavky na tyto funkce jsou klíčové, vytvořený model nebude sloužit jen k prezentaci informací, ale také k dynamické manipulaci s objekty.

R13 - Stromová struktura obsahu reportu – Aplikace umožní zobrazení stromové struktury a obsahu celého reportu. V této struktuře bude možné vybrat požadovanou sekci nebo element pro další úpravu nebo odstranění. Při jakékoliv editaci bude vždy vybrán konkrétní objekt ve stromové struktuře.

R14 – Odezva – Aplikace bude mít ošetřeny veškeré výjimky a nebude zamrzávat. Veškeré chyby budou zobrazeny uživatelům v konzolové komponentě.

R15 - Logovací mechanizmus – Aplikace bude schopná zobrazovat datum a čas tisku, doba potřebná ke zpracování reportu, počet vytištěných řádků.

R16 -Nastavení pozice a velikost jednotlivých elementů – V aplikaci bude možné nastavit pozici a velikosti elementů. Tyto vlastnosti je zapotřebí zprostředkovat s možností úprav myši – především Drag&Drop a změna pozice.

R17 - Ochrana dat – Bude zajištěna ochrana údajů před neoprávněným přístupem. Tento nefunkční požadavek je označen prioritou W a jeho uskutečnění bude provedeno v další verzi aplikace.

R18 - Vizualizace dat pomocí grafů – Pro grafickou prezentaci se budou využívat grafy různých typů (lineární, sloupcové, koláčové, bublinové). Opět tato funkce má prioritu W.

R19 - Vkládání, načítání a ukládání obrázků – V aplikaci bude vytvořeno rozhraní pro práci s obrázky.

3.3.2 Návrh případu užití

ID	Případ užití
UC1	Správa šablon
UC2	Správa obrázku
UC3	Zpracování dat
UC4	Prezentace dat
UC5	Generování tiskové sestavy
UC6	Správa plátna
UC7	Kontrolor
UC8	Paleta elementů
UC9	Navigátor
UC10	Konzole
UC11	Průběžná optimalizace a analýza softwaru
UC12	Výběr dat

Tabulka 5: Jednotlivé případy užití (zdroj: vlastní zpracování)

3.3.3 RTM matice sledovatelnosti požadavků

RTM matice je tabulka, která mapuje jednotlivé systémové požadavky na případy užití. Teprve uvedením systémových požadavků do společného vztahu s případy užití můžeme zjistit, zda požadavky neobsahují něco, co jsme dosud v případech užití nezachytili.

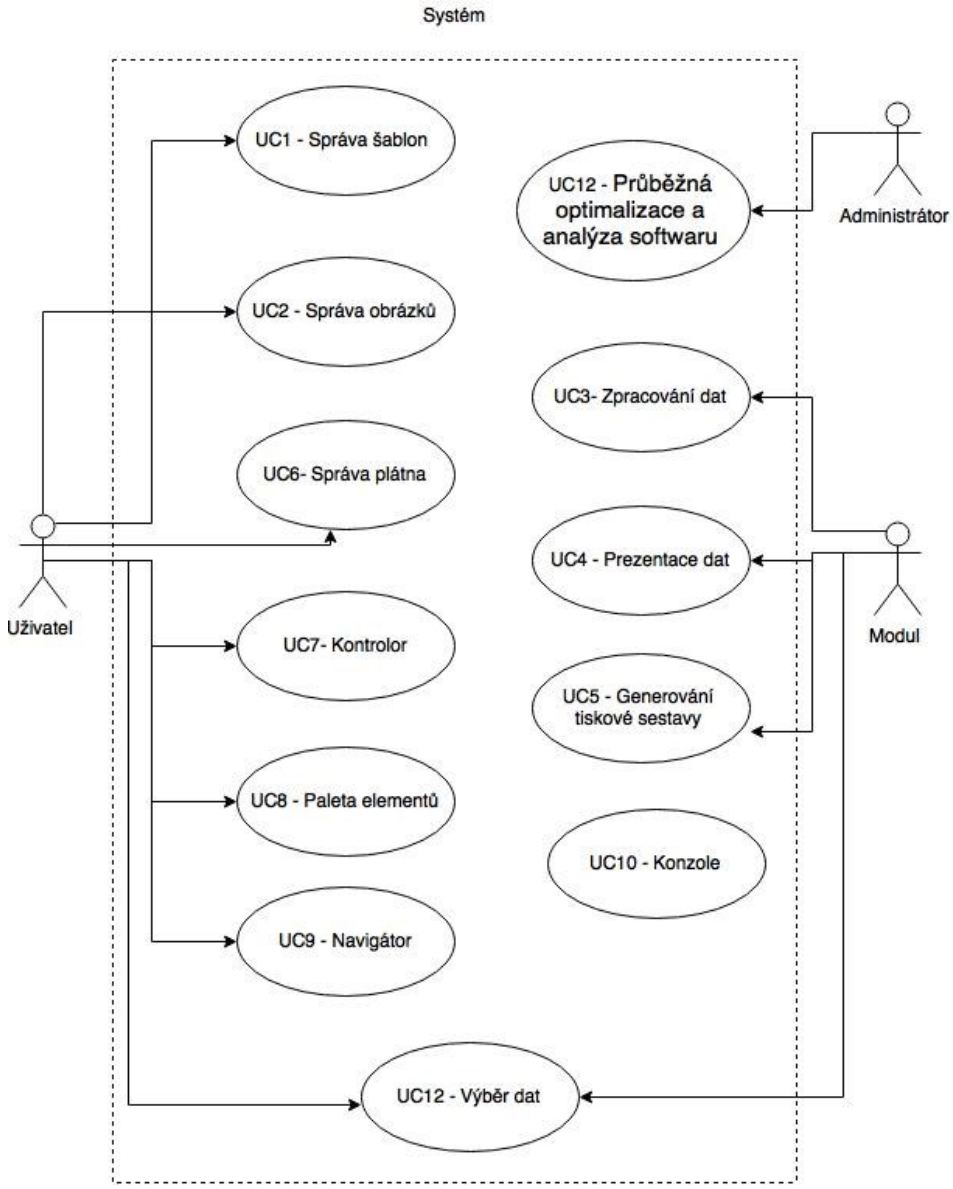
Existuje-li nějaký požadavek, který není mapován na žádný případ užití, znamená to, že v modelu tento případ užití chybí. Toto platí samozřejmě i naopak. (Arlow & Neustadt, 2007) V tabulce číslo 4 nalezneme veškeré požadavky spojené s jednotlivými případy užití.

Uživatelské požadavky	Typy požadavků											
		UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12
	R1												x
	R2			x									
	R3				x								
	R4				x								
	R5								x				
	R6	X											
	R7	X											
	R8								x				
	R9					X							
	R10					X							
	R11											x	
	R12											x	
	R13									x			
	R14											x	
	R15										x		
	R16				x								
	R17											x	
	R18				x								
	R19		x										

Tabulka 6: RTM matice (Zdroj: vlastní zpracování)

3.3.4 Use case diagram

Po předešlých analýzách je možné vytvořit „use case diagram“ (česky diagram případů užití). Tento diagram zobrazuje chování systému tak, jak ho vidí uživatel. Účelem diagramu je popsat funkcionalitu systému. Diagram vypovídá o tom, co má aplikace umět, ale neříká, jak to bude dělat. Je důležité se nejprve shodnout na tom, co má aplikace umět. Až potom má smysl se ptát, jak to vytvoříme.



Obrázek 11:Use case digram - Modul tiskových výstupů (Zdroj: vlastní zpracování)

Výše uvedený diagram popisuje základní funkce a rozhraní, které bude naše webová aplikace obsahovat. Vytvoření těchto funkcí a ovládacích prvků a nástrojů není zdaleka jednoduché. Dynamická práce s objekty je za rozumného úsilí možná až s moderními technologiemi. Díky nim je dnes již možné takové rozhraní vytvořit.

4 Návrh koncepce řešení pro vývoj programového modulu

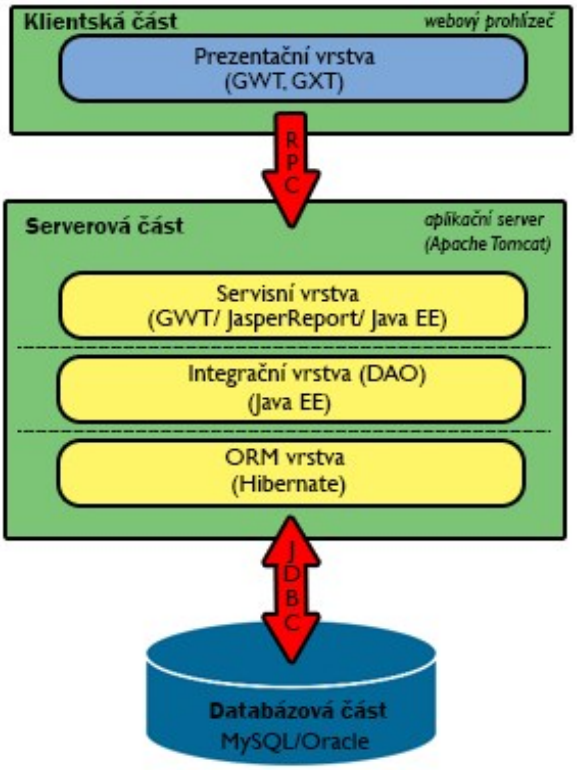
Tato kapitola se zabývá samotným návrhem a implementací webové aplikace podle požadavků, které byly zjištěny v předchozí kapitole. Návrh probírá podle výše uvedených teoretických východisek uvedených v metodologické části této práce. Bylo nutné vybudovat aplikaci tak, aby zahrnovala všechny funkční i nefunkční požadavky. Pro vývoj byly zvoleny technologie představené v metodologických východisek této práce. (Spring Framework, Hibernate, GWT, GXT a JasperReports). Nejprve popíšeme architekturu aplikace.

4.1 Architektura aplikace

Architektura webové aplikace je sestavena typickou třívrstvou architekturou, která obsahuje tři základní části.:

1. Prezentační vrstva (klientská část) – slouží pro zobrazení informací uživateli a interakci s aplikací.
2. Aplikační vrstva (serverová část) – tato vrstva obsahuje hlavní funkce aplikace, je to logická část aplikace.
3. Perzistentní vrstva (databázová část) – zajišťuje správu dat a obstarává přetrvání dat i po ukončení programu.

Každá vrstva má samozřejmě své dílčí části. Například aplikační vrstvu lze rozdělit na další ohraničené vrstvy. Klientská část neboli prezentační vrstva je nejblíže k aplikační vrstvě, která obsluhuje klientskou část a zahrnuje aplikační logiku celé aplikace. Tato servisní vrstva je zcela odloučená od perzistentní vrstvy. Komunikace s perzistentní vrstvou probíhá prostřednictvím integrační vrstvy, která je sestavena z DAO (Data Access Object, objekt pro přístup k datům), sloužící pro manipulaci s daty. Pomocí rozhraní je zajištěn přechod mezi servisní a integrační vrstvou. Přímo za integrační vrstvou je navázaná ORM (Objektově Relační Mapování) vrstva, která zajišťuje automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem za pomocí JDBC. Architekturu aplikace nastiňuje obrázek číslo 12s.



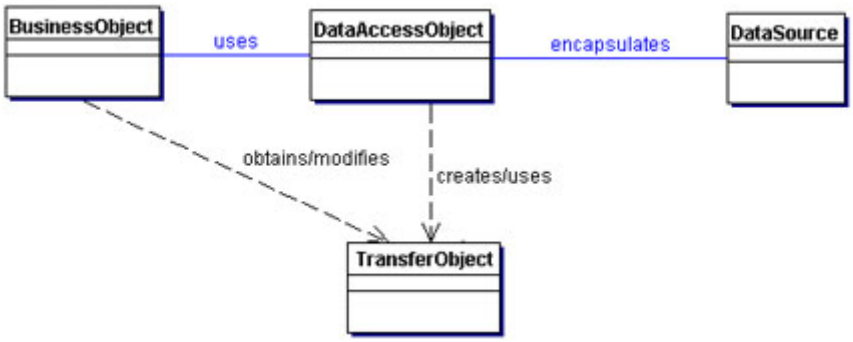
Obrázek 12: Architektura aplikace (Zdroj: vlastní zpracování)

Implementace každé z vrstev aplikace může být zabezpečeno pomocí jiné technologie. V případě našeho projektu je klientská část z programována pomocí frameworku GWT, serverová část pomocí Spring Frameworku a reportovacího nástroje JasperReports a jako nástroj pro objektově-relační mapování aplikace používá Hibernate. Databázová část je realizovaná pomocí relační databází MySQL.

Důležitá je především správná organizace vrstev, musíme zabezpečit, aby se vzájemně příliš neprolínaly. V ideálním případě každá vrstva komunikuje jen s vrstvami sousedními. Tím předejdeme zbytečnému vytváření těsných vazeb mezi jednotlivými komponentami aplikaci. Nyní vysvětlíme objekty typu DAO a DTO a další podkapitolou přejdeme na strukturu aplikace.

4.1.1 DATA ACCESS OBJECT (DAO)

Mezi základní Java EE návrhové vzory patří Data Access Object (objekt pro přístup k datům). Zejména při vytváření webových aplikací. Cílem DAO je izolovat kód pro manipulaci s daty z databáze mimo aplikační vrstvu. V praxi je toho dosaženo tak, že mezi ty části aplikace, které přistupují k datovému zdroji, je vložena DAO vrstva. Ta zahrnuje veškerou logiku pro manipulaci s datovým zdrojem a přístup k němu. Tento přístup znázorňuje obrázek číslo 13, viz níže.



Obrázek 13: Princip DAO (Zdroj: Core J2EE Patterns – Data Access Object, 2015)

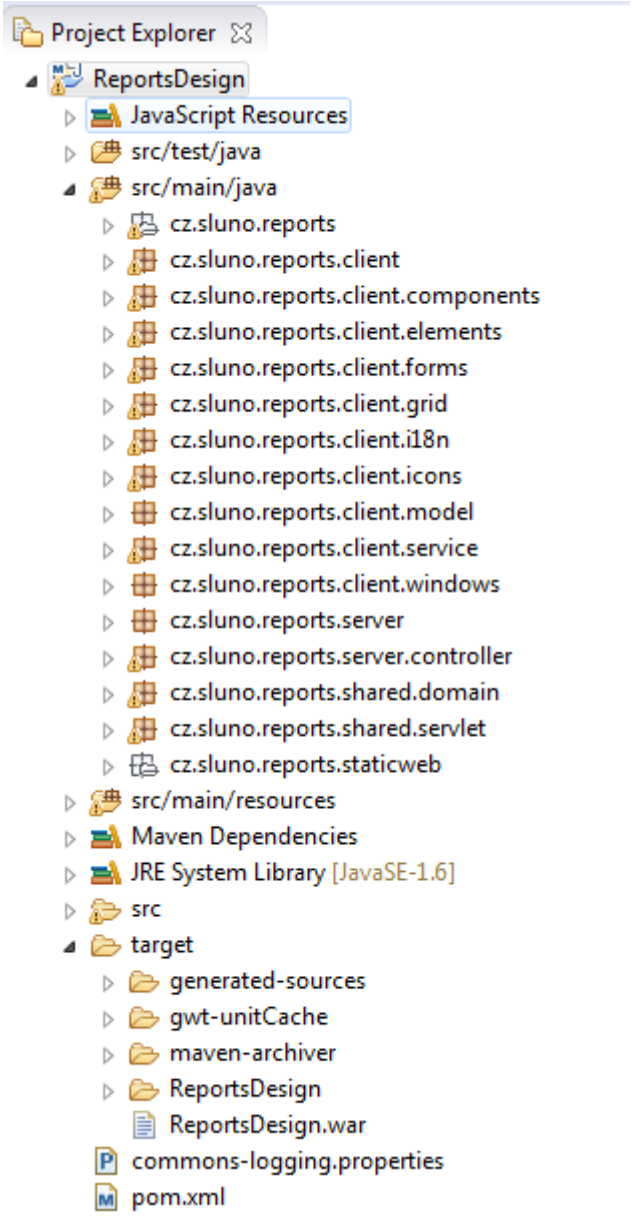
Toto řešení má mnoho výhod. Logika pro práci s daty je přehledně oddělena v jedné vrstvě, což urychluje a usnadňuje její správu a modifikaci. Dále toto řešení zabezpečuje, že se programátor vyhne opakování stejných bloků kódů napříč aplikací a v neposlední řadě je výrazně jednodušší měnit datový zdroj díky tomu, že aplikace je izolovaná od datové vrstvy. Datový zdroj může být relační či objektová databáze, nebo úplně jiný (JSON, XML, atd.).

4.1.2 DATA TRANSFER OBJECT (DTO)

Návrhový vzor Data Transfer Object (objekt pro přenos dat) je definován jako jednoduchý serializovatelný POJO objekt, který obsahuje jednoduché datové pole. Tento objekt neobsahuje žádnou logiku. Daný objekt reprezentuje entitu v databázi. Pomocí tohoto objektu se přenáší data mezi jednotlivými vrstvami aplikace. DTO je používán pro přenos dat mezi DAO a aplikační vrstvou. (Anon., 2015)

4.2 Struktura aplikace

Celá webová aplikace byla vytvořena pomocí zmiňovaného open source modulu Maven, který spravuje projekt a další návaznosti na vývojové prostředí. Maven se postará o stažení všech potřebných knihoven a nástrojů pro vývoj i sestavení aplikace. Na obrázku číslo 14 můžeme vidět strukturu aplikace s názvem „ReportsDesign“.



Obrázek 14: Struktura aplikace (Zdroj: vlastní zpracování)

Celý projekt je tedy, jak je patrné z obrázku zobrazovací jeho strukturu, rozčleněn do tří základních adresářů:

- *src* – zdrojový kód aplikace,
- *Maven Dependecies* – závislé knihovny aplikace,
- *target* – zkompilovaná webová aplikace.

Zdrojový kód je sestaven z celkem 16 balíků doplněných knihovnami, konfiguračními soubory a dále nejrůznějším statickým obsahem. Těchto 16 balíků můžeme rozdělit do čtyř skupin:

- root – kořenový balík,
- client – klientská část aplikace,
- shared – sdílená část aplikace,
- server – serverová část aplikace.

Nastavení GWT projektu, tedy i naší aplikace, je realizováno pomocí konfiguračního souboru XML. Tato konfigurační jednotka definuje, kde je umístěn kompilovaný kód aplikace. Důležité nastavení konfiguračního souboru je upřesnění cesty ke zdrojovým souborům, které jsou převáděny do JavaScriptu, v našem případě je to celý balík client. Také je nutné zvolit vstupní bod aplikace, který udává vstupní metodu pro zobrazení aplikace v prohlížeči. Pomocí konfiguračního souboru můžeme nastavit lokalizaci. Pro každý prohlížeč a danou lokalizaci je kompilována speciální verze aplikace. Nezvyklou vlastností klientské části aplikace je, že komponenty jsou přímo napojeny na objektový model HTML stránky. V praxi to znamená, že manipulace s vlastnostmi komponenty (barva, pozice atd.) dochází přímo v HTML stránce. Úprava jsou ihned viditelná. Musíme počítat s životním cyklem komponent. Některé vlastnosti je možné nastavit až po vykreslení komponent (ne v jejich konstruktoru). GXT komponenty obsahují speciální metodu onRender(), které zabezpečí přepsání a doplnění požadovaných vlastností. Klientské části aplikace nalezneme především komponenty uživatelského rozhraní.

Serverová část obsahující logiku aplikace a veškeré požadavky s klientské vrstvy jsou zprostředkované pomocí asynchronní technologie Ajax. GWT neumožňuje volání synchronních metod, musíme brát ohled na tuto skutečnost. V serverové části nalezneme implementaci služeb na straně serveru a servlety, dále integrační vrstvu z DAO a také z ORM vrstvy obsahující modely a mapovací soubory frameworku Hibernate.

Sdílená část obsahuje soubory, které může využívat jak klientská část, tak i serverová část aplikace. Ve sdílené části nalezneme třídy, které obsahují logické modely aplikace. Nalezneme zde převážně třídy DTO. Nyní se zaměříme na třídy naší aplikace.

4.3 Ukázky implementace aplikace

V následujících odstavcích jsou popsány některé významné principy, části a komponenty webové aplikace a způsob jejích implementace. Nejprve budou popsány konfigurace použitých technologií. Abychom mohli demonstrovat jejich praktické použití. Dále budou popsány návrhové vzory, funkcionality pro návrh tiskových výstupů, jejíž vyřešení patřilo k největším obtížím při vývoji webové aplikace.

4.3.1 GWT konfigurace

Konfigurační XML soubor specifikuje umístění vstupního bodu GWT aplikace, doplňkové komponenty. Dále specifikuje kompilaci kódu a vzhled webové aplikace. Následuje ukázka konfiguračního souboru použitého námi navržené webové aplikace „ReportsDesign“.

```
<?xml version="1.0" encoding="UTF-8"?>
<module rename-to='ReportsDesign'>

    <!-- Inherit the core Web Toolkit stuff.                        -->
    <inherits name='com.google.gwt.user.User'/>
    <inherits name="com.google.gwt.xml.XML" />
    <inherits name="name.pehl.totoe.xml.XML" />
    <!-- inherits library for REST service -->
    <inherits name='org.fusesource.restygwt.RestyGWT' />

    <!-- localization -->
    <inherits name="com.google.gwt.i18n.I18N"/>
    <extend-property name="locale" values="cs"/>

    <!--  inherit css based theme -->
    <inherits name='com.extjs.gxt.ui.GXT'/>
    <inherits name='com.extjs.gxt.charts.Chart' />
    <inherits name='com.extjs.gxt.themes.Themes' />

    <!-- Specify the app entry point class.                         -->
   <entry-point class='cz.sluno.reports.client.Application'/>

    <!-- Specify the application specific style sheet.              -->
    <stylesheet src='Application.css' />
    <!-- Relative package with client source code -->
    <source path='client' />
    <source path='shared'/>

    <!-- Relative package with resources (public static web) -->
    <public path='staticweb' />

</module>
```

4.3.2 Hibernate konfigurace

Konfigurace nástroje pro objektově-relační mapování (Hibernate), je zabezpečena pomocí konfiguračního XML souboru, který dovoluje nastavit chování nástroje a specifikovat autentizační údaje pro přístup k relační databázi. Taktéž specifikuje umístění DTO objektů (viz níže).

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
      <persistence-unit name="MyPUnit"
      transaction-type="RESOURCE_LOCAL">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>

<class>cz.sluno.reports.shared.domain.ImageDTO</class>
<class>cz.sluno.reports.shared.domain.ProductsDTO</class>

<properties>
      <property name="hibernate.show_sql" value="true" />

      <property name="hibernate.c3p0.min_size" value="5" />
      <property name="hibernate.c3p0.max_size" value="20" />
      <property name="hibernate.c3p0.timeout" value="300" />
      <property name="hibernate.c3p0.max_statements" value="50" />
      <property name="hibernate.c3p0.idle_test_period" value="3000" />
      <property name="javax.persistence.jdbc.url"
      value="jdbc:mysql://localhost:3306/sales"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.password" value=""/>
      <property name="javax.persistence.jdbc.driver"
      value="com.mysql.jdbc.Driver"/>
</properties>
</persistence-unit>
</persistence>
```

4.3.3 Spring Framework konfigurace

Existuje několik možností konfigurace hibernate ve Springu, od nejjednodušších, které jsou vhodné pro testování, až po pokročilejší, které jsou vhodné na produkci. Změna konfigurace je pouze změnou v konfiguračním souboru Springu, není nutná žádná změna v kódu aplikace. Ukázku konfigurace beans (objektů spravovaných Springem) nalezneme níže.

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

<context:annotation-config />

<tx:annotation-driven />

<context:component-scan base-package="cz.sluno.reports"/>

<bean class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean"
id="entityManagerFactory">
             <property name="persistenceUnitName" value="MyPUnit" />
</bean>

<bean class="org.springframework.orm.jpa.JpaTransactionManager"
id="transactionManager">
             <property name="entityManagerFactory" ref="entityManagerFactory" />
</bean>
</beans>
```

4.3.4 Implementace návrhového vzoru DAO

Pro přístup k datům v relační databázi se používá integrační vrstva tvořena DAO objekty. Pro všechny objekty přistupující k databázi byla vytvořena abstraktní třída, která obsluhuje základní požadavky pro manipulaci s daty. Její ukázku nalezneme níže:

```
public abstract class AbstractHibernateJpaDAO<K, E> {

       protected Class<E> entityClass;

       @SuppressWarnings("unchecked")
       public AbstractHibernateJpaDAO() {
              ParameterizedType genericSuperclass = (ParameterizedType) getClass()
                           .getGenericSuperclass();
              entityClass = (Class<E>)
genericSuperclass.getActualTypeArguments()[1];
       }

       public void persist(E entity) {
              getEntityManager().persist(entity);
       }

       public void remove(E entity) {
              getEntityManager().remove(entity);
       }

       public void refresh(E entity) {
              getEntityManager().refresh(entity);
       }

       public E merge(E entity) {
              return getEntityManager().merge(entity);
       }

       public E findById(K id) {
```

```
            return getEntityManager().find(entityClass, id);
      }

      public E flush(E entity) {
            getEntityManager().flush();
            return entity;
      }

      @SuppressWarnings("unchecked")
      public List<E> findAll() {
            String queryStr = "SELECT h FROM " + entityClass.getName() + " h";
            Query query = getEntityManager().createQuery(queryStr, entityClass);
            List<E> resultList = query.getResultList();
            return resultList;
      }

      public Integer removeAll() {
            String queryStr = "DELETE FROM " + entityClass.getName() + " h";
            Query query = getEntityManager().createQuery(queryStr);
            return query.executeUpdate();
      }

      protected abstract EntityManager getEntityManager();
}
```

Díky této abstraktní třídě stačí pouze, aby naše DAO třídy dědily tyto třídu a přepsaly metodu getEntityManager(); Ukázka přepsání abstraktní třídy:

```
@Repository("imageDAO")
public class ImageDAO extends AbstractHibernateJpaDAO<Long, ImageDTO>{
      @PersistenceContext(unitName = "MyPUnit")
      EntityManager entityManager;

      @Override
      protected EntityManager getEntityManager() {
            // TODO Auto-generated method stub
            return entityManager;
      }


}
```

4.3.5 Implementace návrhového vzoru DTO

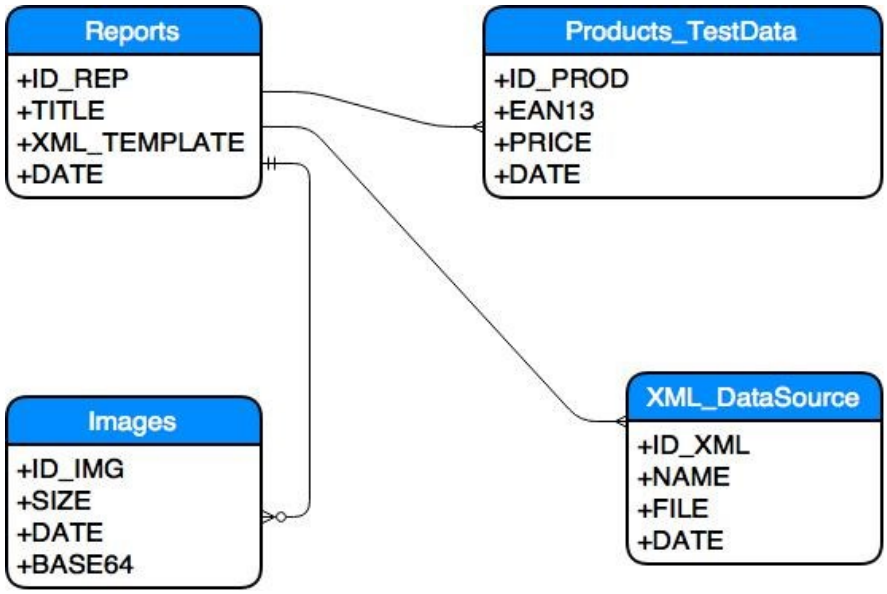
Pro přenos dat v aplikaci je použit vzor DTO. Pro položku obrázky je následující ukázka DTO (ImageDTO) :

```
public class ImageDTO implements Serializable, BeanModelTag {
      @Id
      @GeneratedValue(strategy = GenerationType.AUTO)
      @Column(name = "id_img")
      private Long id;

      private String image_name;
      private byte[] content;
```

```
private String description;

@Transient
private String base64;

public ImageDTO() {
            }


public ImageDTO(ImageDTO image) {
      this.id = image.getId();
      this.image_name = image.getImage_name();
      this.content = image.getContent();
      this.description = image.getDescription();
      this.base64 = image.getBase64();
}
...
//Getters/setters
```

4.3.6 ER diagram datového modelu

ER diagram je první obecný pohled návrhu databáze na zúčastněné entity a vztahy mezi nimi. Musíme poznamenat, že diagram není úplný. Společnost U&SLUNO nezpřístupnila informace ohledně jejich databázových systémů z důvodu bezpečnosti.



Obrázek 15: ER diagram (Zdroj: vlastní zpracování)

Tabulka „Reports“ slouží pro ukládání reportovacích šablon a jejich opětovného načtení.

Tabulka „Images“ slouží pro správu obrázků, které může report obsahovat. Obrázky jsou v databázi uložené v kódovacím algoritmu „base64“.

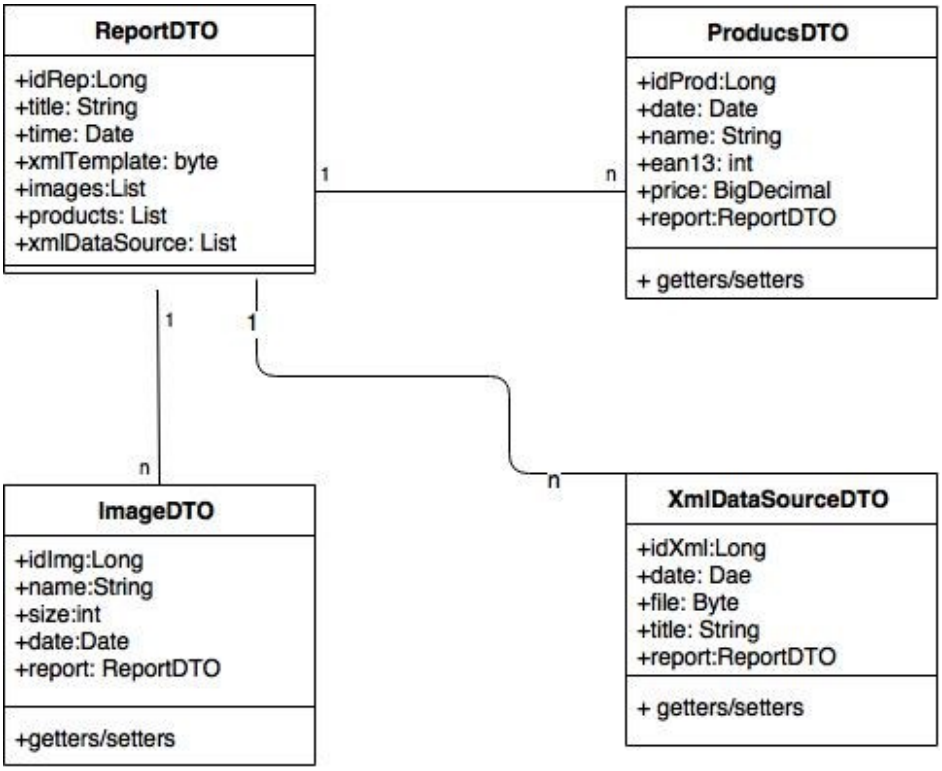
Tabulka „Products_TestData“ obsahuje reálné data poskytnuté společností U&SLUNO. Data popisují katalog produktů pro testování generování čárových kódu.

Tabulka „XML_DataSource“ obsahuje údaje o XML souborech, které mohou být použity jako datový zdroj pro generování reportů.

Diagram tříd DTO

Diagram tříd DTO reprezentuje datový model v databázi. Pomocí nástroje ORM v našem případě Hibernate. Můžeme vygenerovat celou strukturu databázového modelu do relační databáze pouze přidáním příkazu:

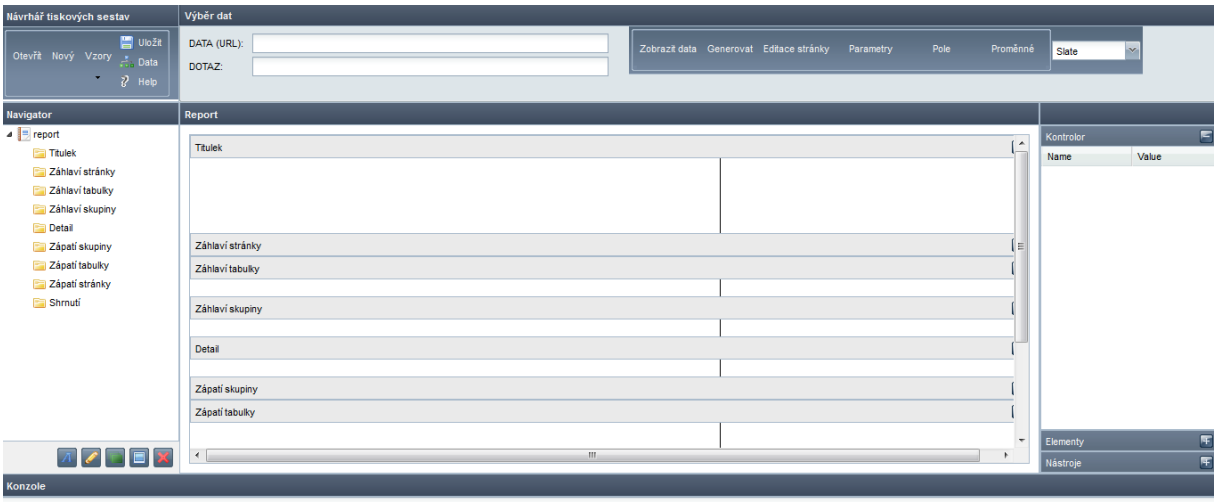
`<property name="hibernate.hbm2ddl.auto" value="create" />` do konfiguračního soboru hibernate.



Obrázek 16: Diagram tříd DTO (zdroj: vlastní zpracování)

4.4 Vzhled uživatelského rozhraní aplikace

Uživatelské rozhraní je jednou z nejdůležitějších částí webové aplikace. Při realizaci uživatelského rozhraní byl kladen důraz na přehledné a jednoduché ovládaní. Ovládací prvky byly navrženy s ohledem na rozdílné velikosti zobrazovacích zařízení. Aplikace se automaticky přizpůsobí velikosti obrazovky pomocí responzivního návrhu. GWT je založeno na způsobu kontejnerů. Do jednotlivých kontejneru můžeme vkládat komponenty, které jsou automaticky zarovnávány dle vybraného způsobu zarovnávání. V našem případě bylo použito rozložení BorderLayout, který svůj obsah rozděluje na světové strany.



Obrázek 17: Grafický návrh uživatelského rozhraní (Zdroj: vlastní zpracování)

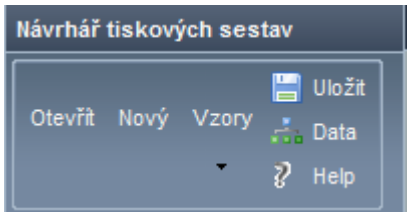
V horní části obrazovky můžeme vidět správu šablon a rozhraní pro výběr dat. Ve středu obrazovky je reportovácí plátno, které slouží pro rychlou úpravu pozice, velikosti elementů. Vlevo od repotovacího plátna se nachází navigátor a vpravo kontrolor, pro detailní nastavení vlastností elementů. Ve spodní části je umístěna konzole, která zobrazuje generování chyb v průběhu generování tiskového výstupu. Nyní se zaměříme na jednotlivé komponenty.

4.4.1 Správa tiskových šablon

Správa tiskových šablon obsahuje funkce, které jsou určené pro vytvoření nové prázdné šablony, ukládání rozpracované šablony nebo otevření šablony ze souboru.

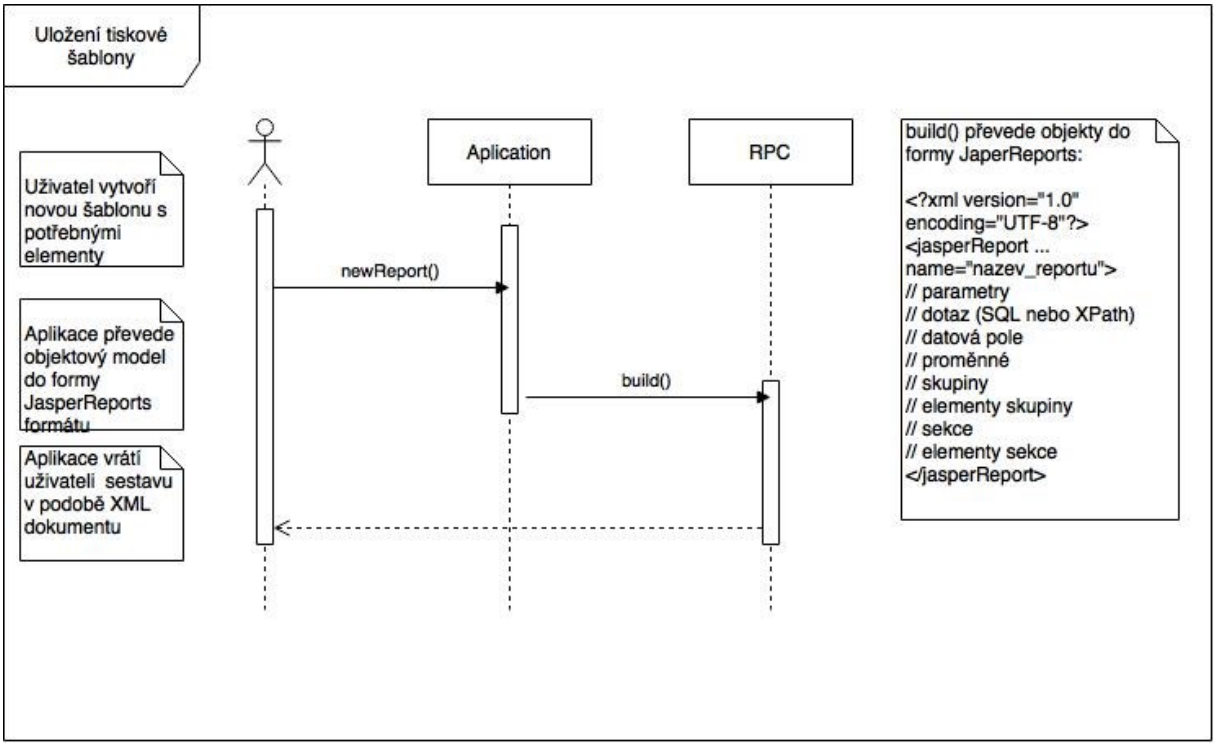
Funkce pro uložení šablony probíhá převedením webových elementů do nativního kódu JasperReportu, který se uloží na lokální disk v XML formátu.

Funkce načítání šablony poskytne vybrat lokální soubor ve formátu JaspeReports XML a převést ho do objektového modelu GWT a výsledná tisková sestava se zobrazí v uživatelském rozhraní. Uživatelské rozhraní pro správu šablon ilustruje obrázek číslo 18.



Obrázek 18: Uživatelské rozhraní pro správu tiskových šablon (Zdroj: vlastní zpracování)

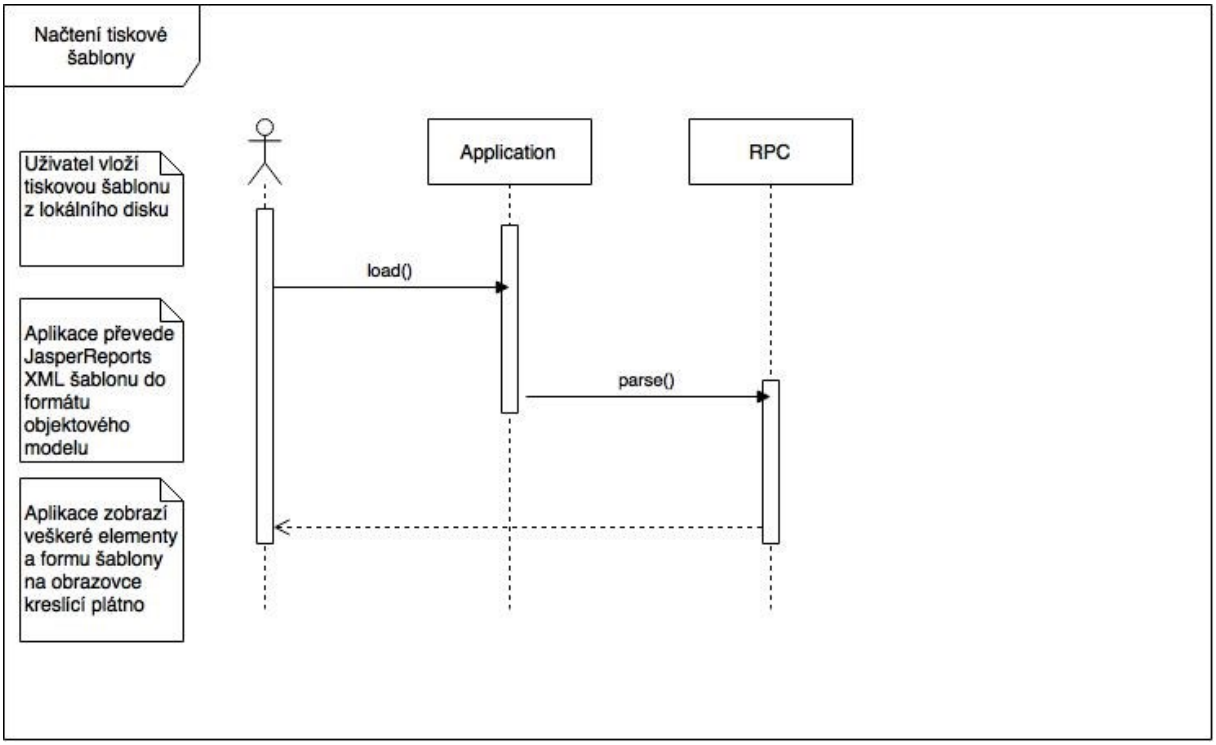
Obrázek číslo 19. popisuje sekvenční diagram pro ukládání tiskové šablony. Hlavní metoda build() realizuje sestavení XML souboru na základě modelu sekcí a elementů.



Obrázek 19: Sekvenční diagram ukládání šablony (Zdroj: vlastní zpracování)

Načtení tiskové šablony je realizováno metodou parse(), která slouží pro sestavení objektového modelu a následném zobrazení šablony v uživatelském rozhraní „kreslící plátno“. Úplný proces se skládá z načtení obecných vlastností tiskové šablony, seznam datových polí, parametrů, proměnných, skupin sekcí a konkrétních elementů (text, textové pole, čára, obdélník, obrázek, čárový kód atd.)

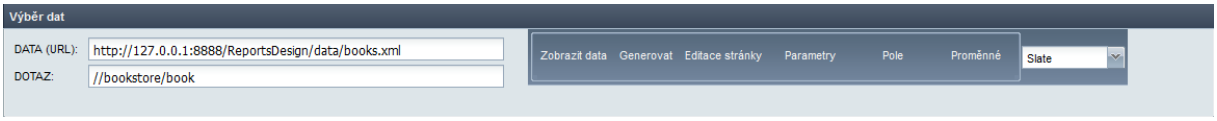
Načtení tiskové šablony je realizováno tak, aby v případě nepodporované funkce či elementu jejich načtení bylo automaticky přeskočeno. To je velmi užitečné, zvlášť budeme-li chtít do aplikace nahrát tiskovou šablonu vytvořenou ve variantě desktopového návrháře. (viz obrázek 20)



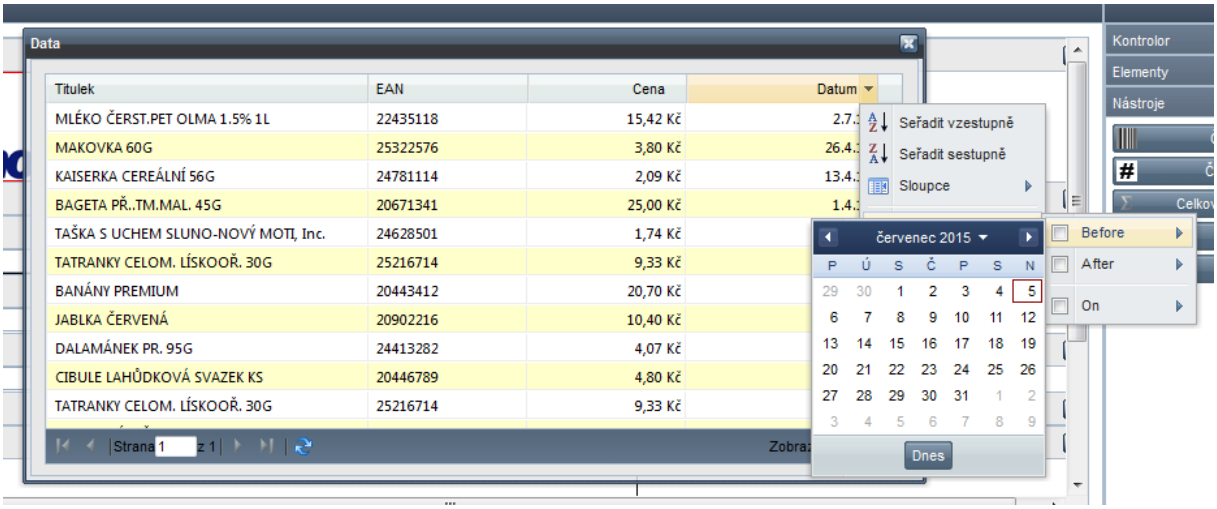
Obrázek 20: Sekvenční diagram načtení šablony (Zdroj: vlastní zpracování)

4.4.2 Výběr dat

Výběr dat dovoluje definovat datový zdroj XML formátu, tlačítko zobrazit data poskytuje otevření nové záložky v prohlížeči pro zobrazení dat. Panel pro správu dat ilustruje obrázek číslo 21. Dále je možné využít uživatelské rozhraní pro práci s daty z relační databáze, toto rozraní umožňuje data filtrovat a seřazovat. (viz obrázek číslo 22).



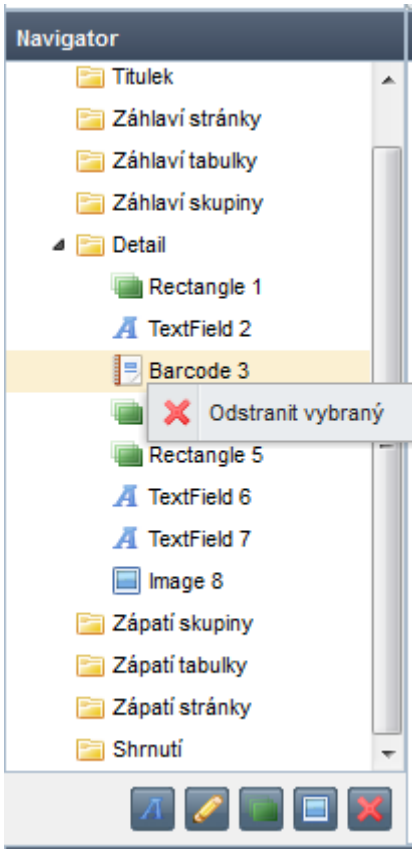
Obrázek 21: Panel pro správu dat (Zdroj: vlastní zpracování)



Obrázek 22: Uživatelské rozhraní pro správu dat (Zdroj: vlastní zpracování)

4.4.3 Navigátor

Navigátor slouží pro zobrazení stromové struktury celého reportu. Pomocí navigátoru můžeme snadno odstranit vybraný element a to díky kontextového menu. Dále můžeme snadno vybrat element nebo požadovanou sekci pro další úpravy (viz níže).



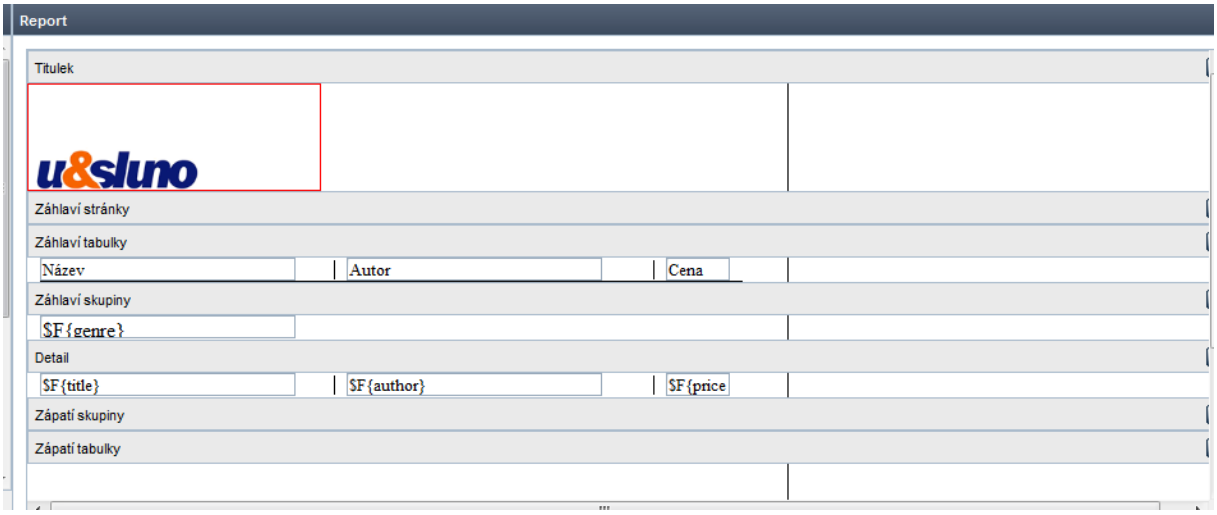
Obrázek 23: Stromová struktura reportu (Zdroj: vlastní zpracování)

Dále na navigátoru je závislý kontrolor, který slouží pro zobrazování vlastností vybraného elementu nebo sekce v reportu.

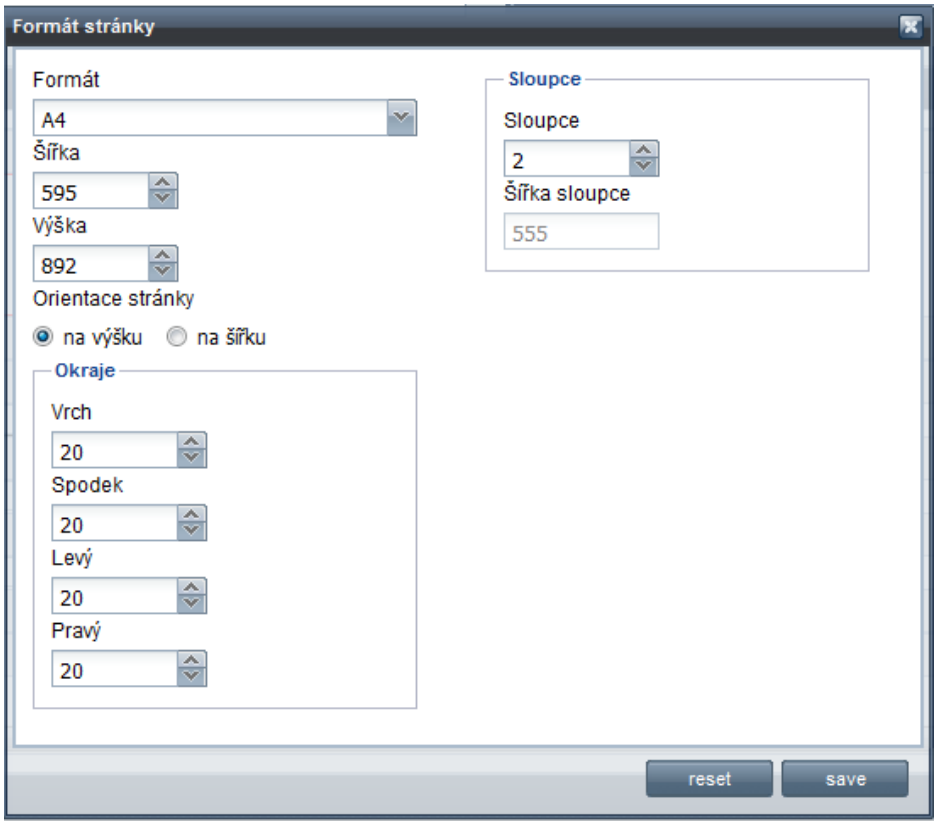
4.4.4 Prezentace dat

Pro prezentaci dat slouží kreslící plátno (obrázek číslo 24), které je rozděleno podle sekcí JasperReportu. Každá sekce umožňuje uživateli nastavit její velikost. K tomu je využita komponenta Sencha GXT – BorderLayout, která dovoluje nastavit maximální a minimální velikost dané sekce. V případě změny velikosti dané sekce, webová aplikace automaticky přepočítá zbývající sekce. Pro přesnější nastavení reportu můžeme použít vytvořený formulář „Formát stránky“, kterou ilustruje obrázek číslo 25. Pomocí tohoto formuláře můžeme nastavit okraje stránky, orientaci (na šířku, na výšku) nebo dokonce rozdělit reportu na několik sloupců. Ve výchozím nastavení má report pouze jeden sloupec.

Kreslicí plátno nám dále dovoluje přímou změnu velikosti a pozice elementu za pomocí počítačové myši. Různé vygenerované sestavy nalezneme v číslo 2.



Obrázek 24: Kreslící plátno webové aplikace

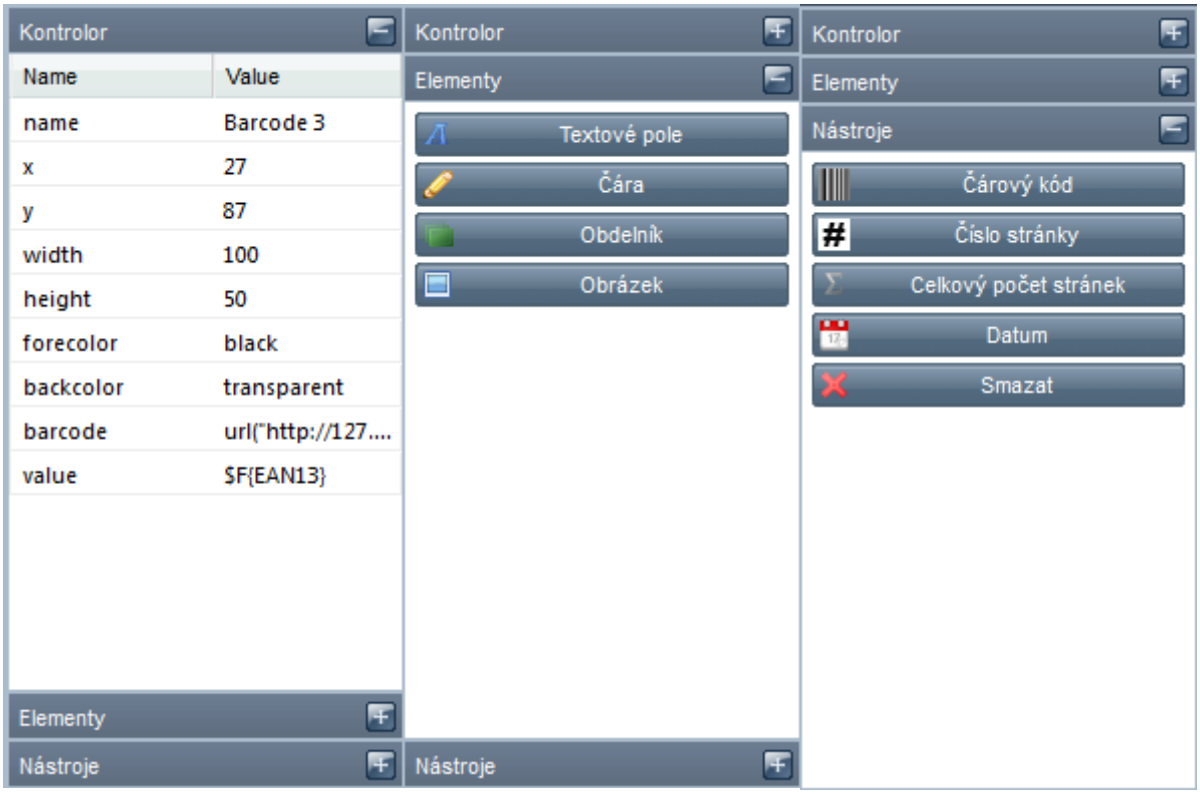


Obrázek 25: Formulář formát stránky (Zdroj: vlastní zpracování)

4.4.5 Kontrolor

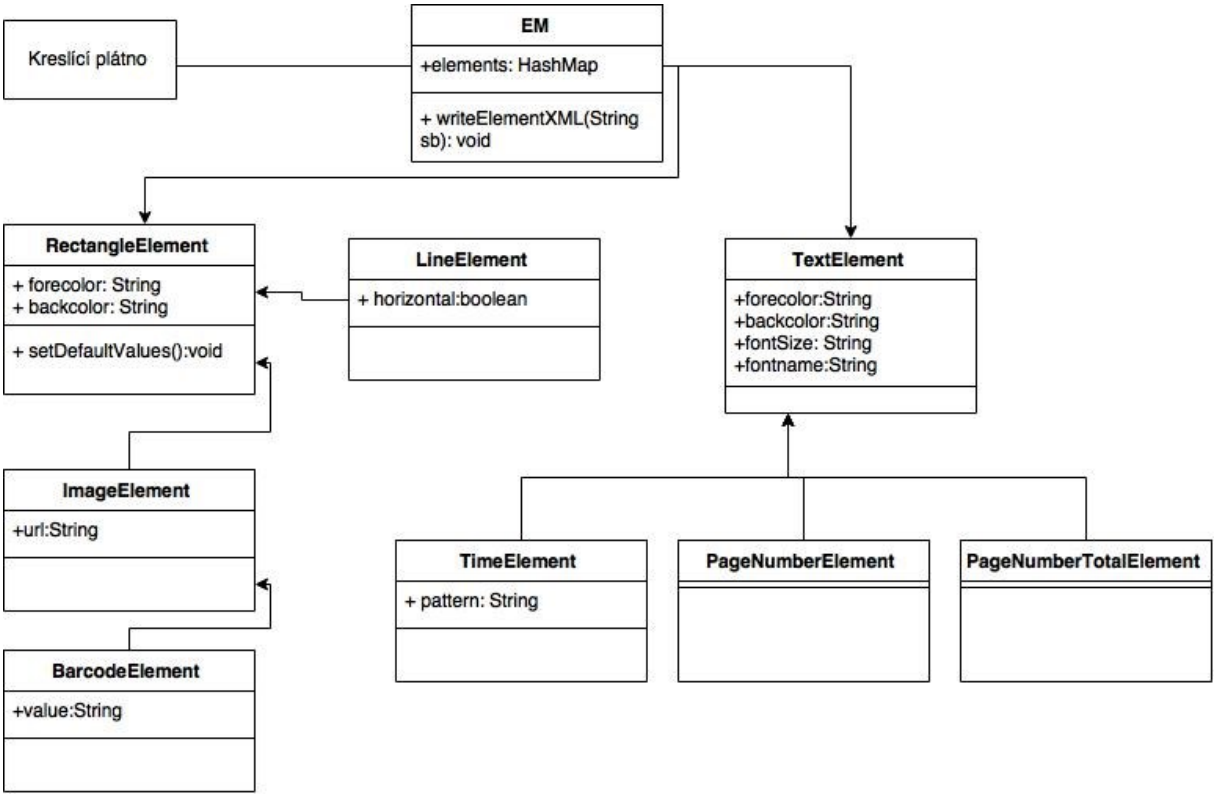
Kontrolor slouží k úpravě specifických vlastnosti elementů či sekcí. Kontrolor byl vytvořen pomocí komponenty Sencha GXT – **AccordionLayoutContainer,** který spravuje více obsahu panelu v harmonikovém stylu. Pouze jeden panel může být rozšířen v daném času. V našem případě jsou to panely:

- Kontrolor,
- Elementy – Textové pole, čára, obdélník, obrázek.
- Nástroje – čárový kód, číslo stránky, celkový počet stránek, datum, smazat.



Obrázek 26: AccordionLayoutContainer - Kontrolor, Elementy, Nástroje (Zdroj: vlastní zpracování)

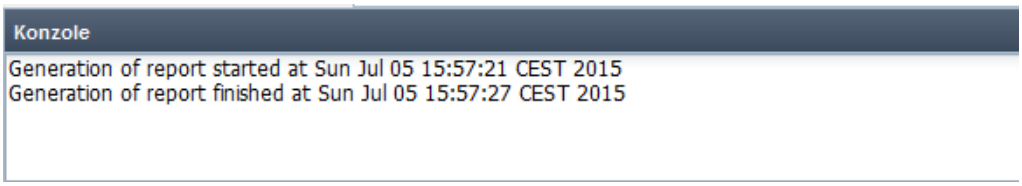
Kreslicí plátno využívá elementy a nástroje jako hlavní prvky pro generování tiskových výstupů. Diagram tříd znázorňuje obrázek číslo 27.



Obrázek 27: Diagram tříd elementů (Zdroj: vlastní zpracování)

4.4.6 Chybová Konzole

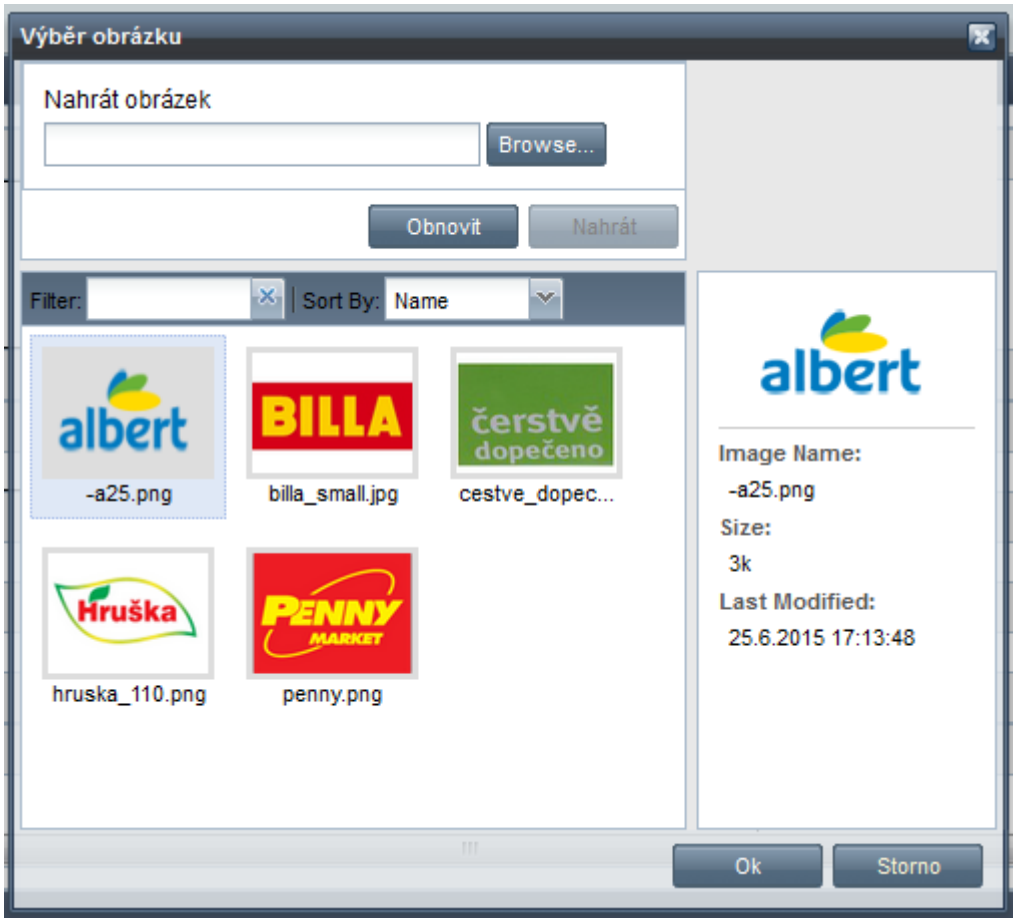
Chybová konzole se nachází ve spodní části aplikace a slouží k upozorňování chyb, které nastanou při generovaní tiskových výstupů (obrázek číslo 28).



Obrázek 28: Chybová konzole (Zdroj: vlastní zpracování)

4.4.7 Správa obrázků

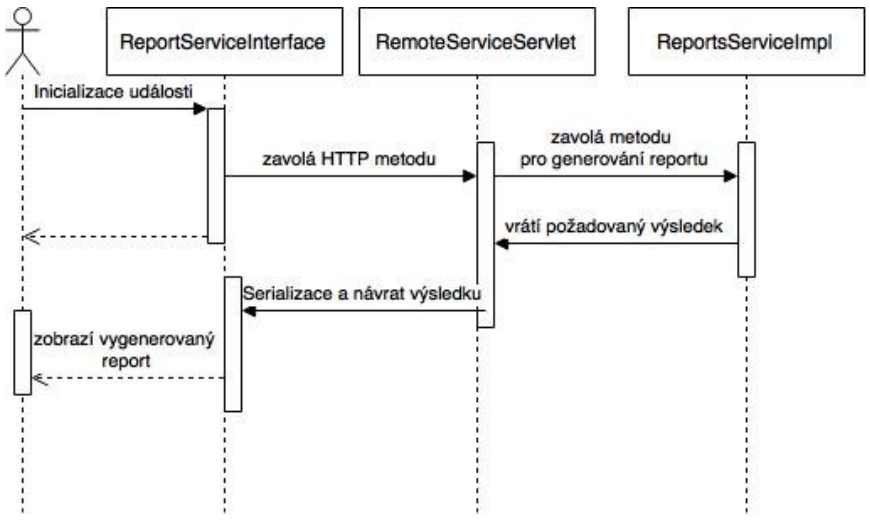
Další uživatelské rozhraní, které bylo zapotřebí vytvořit je správa obrázku. Správa obrázku dovoluje vybrat obrázek do dané sekce. Dále umožňuje filtrovat a řadit obrázky podle jména, velikosti a datumů nahrání obrázku. Uživatelské rozhraní pro správu obrázku ilustruje obrázek číslo 29.



Obrázek 29: Uživatelské rozhraní správa obrázku (Zdroj: vlastní zpracování)

Nyní se zaměříme na funkci generování tiskových výstupů podle sekvenčního diagramu.

4.5 Sekvenční diagram pro generování tiskového výstupu



Obrázek 30: Sekvenční diagram pro generování tiskového výstupu

Pro tvorbu tiskového výstupu potřebuje šablonu JRXML formátu JasperReports. Do naši aplikace je vložena podpora výstupu PDF, HTML a XML. Tato funkce je dostupná prostřednictvím GWT RPC, jak můžeme vidět výše uvedeného sekvenčního diagramu.

V případě, že generování tiskového výstupu proběhne úspěšně, je výsledný binární soubor zaslán zpět uživateli do nové záložky prohlížeče. Možnosti vygenerovaných reportů nalezneme v příloze číslo 2.

5 Použití certifikátů k odesílání šifrovaných zpráv

Bohužel nebylo provedeno použití certifikátů pro odesílání zpráv, jelikož implementace webové aplikace se neuskuteční v termínu odevzdání této diplomové práce. O použití certifikátu se uvažuje v dalších verzích aplikace.

6 Zhodnocení navržené koncepce

Pro vývoj webové aplikace byla zvolena platforma Java EE a při něm využita kombinace frameworků GWT, Sencha GXT, Hibernate, Spring a JasperReports. Zkušenosti a poznatky nabyté jejich studiem byly představeny průběžně v textu této diplomové práce. V následujících odstavcích budou sumarizovány a zobecněny.

Kombinace GWT a Sencha GXT významnou měrou zjednodušila a zrychlila tvorbu webové aplikace. Díky tomuto spojení můžeme vyvíjet aplikace bez znalostí JavaScriptu a specifikacích webových aplikací, pomocí nástrojů platformy Java. Pomocí tohoto nástroje je programátor odstíněn od typických problémů spojené s tvorbou webu, jako je rozraní lokálních a serverových částí aplikace při serializace a deserializace objektů při komunikaci. Avšak použití tohoto nástroje vyžaduje specifický přístup při vývoji, který je založen pouze na asynchronním volání logicky na straně serveru. Další nevýhodou je, že automatický převod z jazyka Java do JavaScript pro všechny podporované prohlížeče je výpočetně náročný úkol[2]. To vede ke zdlouhavému vývoji. Tento problém částečně řeší vývojový mód, kdy nedochází k hromadnému převodu do JavaScriptu.

Největší výhodou je použití frameworku Sencha GXT, která obsahuje širokou paletu předdefinovaných komponent, které jsou snadno rozšiřovatelné a upravovatelné. To sebou nese úskalí, a sice že webové aplikace mají v praxi často uniformní vzhled. V našem případě byly použity tři grafické vzhledy, které můžeme měnit za chodu aplikace. Grafické návrhy nalezneme v příloze číslo 1.

Další výhodou GWT je dostupnost integrovaných vývojových prostředí (IDE) jako je Eclipse, NetBeans a jiné. Za pomocí těchto prostředí mohou vývojáři využívat veškeré oblíbené nástroje pro vývoj webových aplikací. Vzhledem k tomu, že za GWT stojí silná vývojářská komunita, která napomáhá k usnadnění problémů při vývoji, je tvorba webových aplikací o dost jednodušší.

Další použitý Framework Hibernate (nástroj pro ORM), který usnadňuje vývoj webových aplikací v oblasti problémů spojených s perzistencí dat. Výhodou tohoto nástroje je především flexibilita a to zejména v oblasti dotazování (HQL, QBC, nativní SQL), a také při řízení životního cyklu objektů.

[2] Na počítači s dvoujádrovým procesorem Intel Core i5 s frekvencí 2.40 Ghz a 6GB RAM trvala kompilace 4 minuty.

Pro programátory, které nemají zkušenosti s ORM, je počáteční přístup obtížný a většinou jim zabere určitý čas s jeho seznámením. Zejména pří využívání návrhových vzorů (v našem případě DAO), avšak při jejich použití mnozí ocení možnost reprezentaci entit z relačních databázích v podobě POJO objektů. Příjemnou výhodou je dostupnost podpůrných nástrojů, například použití zásuvného modulu Hibernate Tools, které automatizuje řadu procesů, jako je generování entit z databáze.

Spring Framework poskytuje silnou infrastrukturní podporu a dále je to dosti robustní nástroj. Velkou výhodou je jeho modularita a vývojář používá jen potřebné zvolené součásti. Při vývoji naši aplikace byly využity jen jeho základní funkce. Pro lepší pochopení funkcí a možností, které Spring nabízí, by bylo zapotřební mnohem delší studium.

Společné vlastnosti všech frameworků, které byly použity je otevřenost dokumentace a hlavně velké podpory ze strany komunity vývojářů. Dále velké množství návodů a také ukázky již vyřešených a často se vyskytujících problémů. Velkým problémem z počátku může být právě správná počáteční konfigurace pro zabezpečení integrace. Další problémy mohou být správné ladění kód, protože chyby probublávají různými komponentami aplikace, někdy je složité dohledat příčinu vzniklé chyby. Dobrým základem je seznámení se nejprve s jednotlivými frameworky postupně, či strávit nějaký čas experimenty. Po překonání počátečních problémů můžeme uznat, že práce s nimi byla efektivní a postupem času i pohodlná.

6.1 Zhodnocení etapy realizace webové aplikace

Realizace webové služby probíhala podle zmíněného vodopádového modelu. Jakkoliv se funkce zdají být předem jednoduché, ukázalo se jako velmi dobré, je předem formálně definovat. Tímto se tak zvýšila šance, že budou všechny funkce dodrženy.

Velkou snahou autora při tvorbě aplikace bylo oddělení etapy návrhu od samotné realizace. To hlavně z důvodu, aby bylo přesně jasné, co je třeba udělat, a tak předejít zbytečné tvorbě komponent, které nakonec nebudou použity. To lze dosáhnout jedině důkladnou analýzou, správným vyhodnocením všech informací a pečlivým přístupem k procesu návrhu webové aplikace. Domnívám se, že je hlavní činností při realizaci jakékoliv aplikace právě návrhová část. Důležité je dořešení všech částí a funkcionalit, které má aplikace obsahovat. Nakonec právě takový přístup je časově efektivnější, než se zaobírat funkcionalitami až ve chvíli, kdy aplikace už má vyvinuté jiné části softwaru.

Celkové můžeme označit etapu realizace webové aplikace jako úspěšnou a efektivní. Od začátku byl kladen důraz na uživatelskou přívětivost a byly vybrány jen nejpoužívanější funkce. Odvažuji se tvrdit, že komfort uživatelského rozhraní je srovnatelný s desktopovými řešení a to je pro webovou aplikaci obrovský úspěch. Praktickou část této práce můžeme označit jako jejich alternativu. Nicméně není spravedlivé srovnávat námi vytvořené řešení s desktopovými variantami, které mají delší než desetiletý vývoj a mají o mnoho více funkcí. Vývoj probíhal systematicky podle metodik softwarového inženýrství. Kromě několika úprav a přepracování zadání byl také plynulý.

7 ZÁVĚR

Uvedením webové aplikace jako modul pro generování tiskových výstupů, kdy pomocí ní lze provádět mnohé návrhy pro tisk, byl naplněn hlavní cíl této diplomové práce. Ideální cesta k naplnění cíle spočívá v naplnění několika dílčích cílů.

První dílčí cíl spočíval ve vytvoření návrhu webové aplikace, které byl spojen důkladnou analýzou funkčních i nefunkčních požadavků. Výsledky těchto analýz byly zohledněny při návrhu webové aplikace.

Druhým dílčím cílem bylo zvolení vhodných technologií, nástrojů a postupů pro implementaci webové aplikace. Ty byly zváženy právě z analýzy požadavků. Pro samotný vývoj webové apliace byla vybraná platforma Java EE a kombinace Spring, Hibernate, GWT, GXT a JasperReports frameworků. Klíčové poznatky, které autor nabyl při vývoji aplikace, byly v práci prezentovány.

Třetím dílčím cílem byla samotná realizace webové aplikace. Při této fázi se ukázalo, že některé zvolené technologie a postupy byly pro dosažení hlavního cíle až zbytečně složité a robustní. To vedlo k prodloužení celého vývoje. Na druhé straně po překonání počátečních problémů spojené se seznamováním technologií už vývoj probíhal rychle a efektivně. Dle mého názoru použití vybraných frameworků přineslo přehlednost zdrojového kódu aplikace a její budoucí údržbu.

Poslední čtvrtý dílčí cíl spočívající ve spuštění webové aplikace a předvedení její základní funkčnosti byl bohužel splněn jen částečně. Webová aplikace je sice připravená k okamžitému spuštění, nebyla však nasazena do ostrého provozu. Avšak byla použita testovací data, katalog reálných produktů pro generování tiskových etiket na zboží. Data byla poskytnutá společností U&SLUNO a. s. viz (http://www.u-sluno.eu/). Ověření požadované funkčnosti probíhalo na lokálním serveru. Ukázka webové aplikace je přiřazena mezí přílohy práce, spolu s daty a podklady pro simulaci provozu.

Co se týče praktického přínosu diplomové práce jako webové aplikace, slouží pro generování reportů do různých formátů. Hlavní teoretický přínos je poskytnutí strukturované dokumentace, jak navrhnout a vytvořit webovou aplikaci, která je zaměřena na specifické potřeby.

Dalším přínosem této práce, které poskytnulo její vypracování pro autora, je zejména seznámení se s novými technologiemi pro tvorbu webových aplikací. Dále pak zlepšování analytických dovedností získaných při modelování procesů a navrhování softwaru.

Webová aplikace jako „Programový modul pro návrh tiskových výstupu“ byla úspěšně vytvořena a je připravena k nasazení. Abychom mohli využít naplno její potenciál, tak musíme upravit aplikaci v oblastech bezpečnosti. Právě použití certifikátu třetí stranou k odesílání šifrovaných zpráv mezi informačními systémy může výrazně zvýšit bezpečnost aplikace. Poukazujeme na to, že stále je velký prostor na napojení mnoha dalších funkcí, jako jsou grafy, křížové tabulky či subreporty. Další úpravou může být rozšířená lokalizace. V současném stavu má aplikace dva překlady a to jazyk český a anglický – ukázky nalezneme v příloze 1. Tato a případně další úpravy pro rozvoje aplikace bude značně záviset na úspěchu při reálném použití.

Seznam použité literatury

Odborná literatura

Arlow, J. & Neustadt. UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

BASHAM, Bryan, Kathy SIERRA a Bert BATES. Head first servlets and JSP. 2nd ed. Sebastopol, [Calif.]: O'Reilly, 2008. ISBN 978-059-6516-680.

Bauer, C. & King, G. Java persistence with Hibernate. Greenwich: Manning Publications, 2015. ISBN 978-1617290459.

Buchalcevová, A., Pavlíčková, J. & Pavlíček, L. Základy softwarového inženýrství - materiály ke cvičení. Praha: Oeconomica, 2007. ISBN 978-80-245-1270-9.

Danciu, T. & Chirita, L. The definitive guide to JasperReports. Berkeley, CA: Apress, 2007. ISBN 15-905-9927-6.

Heffelfinger, D. R.. JasperReports 3.5 for Java developers. Birmingham [u.a.]: Packt Publ, 2009. ISBN 978-1847198082.

KADLEC, V. Agilní programování: Metodiky efektivního vývoje softwaru. Brno: Computer Press, 2004. ISBN 80-251-0342-0.

Kristián, P. FrontPage 2000 a návrh webu. Brno: UNIS Publishing, 2001.ISBN 80-860-9757-9.

Ponkrác, M. PHP a MySQL: bez předchozích znalostí. Brno: Computer Press, 2007. ISBN 978-80-251-1758-3.

Resig, J. JavaScript a Ajax: moderní programování webových aplikací. Brno: Computer Press, 2007. ISBN 978-80-251-1824-5.

Siddiqui, B. JasperReports 3.6 development cookbook. Birmingham, United Kingdom: Packt Publishing, 2010. ISBN 978-1-84951-077-6.

Stephens, R. Beginning software engineering. Indianapolis: John Wiley & Sons, Inc., 2015. ISBN 978-1118969144.

Tacy, A., Hanson, R., Essington, J. & Tökke, A. GWT in Action: SECOND EDITION. Shelter Island, NY: Manning Publications Co., 2013. ISBN 978-193-5182-849.

Varanasi, B. & Belida, S. Introducing Maven. New York: Apress Media, 2014. ISBN 978-1484208427.

Vondrák, I. Úvod do softwarového inženýrství. Skripta VŠB-TU, Ostrava, 2002.

Walls, C. Spring in action. Shelter Island, NY: Manning Publications Co., 2013. ISBN 978-1617291203.

Wiegers, K. E., 2008. Požadavky na software. Brno: Computer Press, 2008. ISBN 978-80-251-1877-1

Elektronické dokumenty

Core J2EE: Core J2EE Patterns - Data Access Object. Core J2EE.com [online]. 2015 [cit. 2015-04-22]. Dostupné z: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html

Garrett, Jesse. Ajax: A new Approach to Web Applications. adaptivepath.com [online]. 2015 [cit. 2015-02-18] Dostupné z: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/

managementmania.com. [online]. 2015 [cit. 2015-04-15] Dostupné z: https://managementmania.com/cs/webova-aplikace-web-application

Sencha GXT: Create feature-rich HTML5 applications using Java and Google Web Toolkit. Sencha.com [online]. 2015 [cit. 2015-04-21]. Dostupné z: http://www.sencha.com/products/gxt/#overview

Seznam zkratek

Ajax	Asynchronous JavaScript and XML
AOP	Aspect-oriented Programming
API	Application Programming Interface
atd.	a tak dále
CASE	*Computer-aided sotware engineering*
CSS	Cascading Style Sheets
CSV	Comma-separated values
DAO	Data Access Object
DARPA	Defense Advanced Research Agency
DTO	Data Transfer Object
ER diagram	Entity-relationship diagram
GWT	Google Web Toolkit
HQL	Hibernate Query Language
HTML	HyperText Markup Language
ID	identifikační číslo
IoC	Inversion of Control
JAVA EE	Java Enterprise Edition
JDBC	Java Database Connectivity
JPA	Java Persistance API
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LGPL	Library General Public License
MVC	Model - view - controller
např.	Například
ODT	OpenDOcument
ORM	objektově-relační mapování
PDF	Portable Document Format
POJO	Plain Old Java Object
PPTX	specifikace Office Open XML
QCB	Query By Criteria

RIA	Rich Internet Applications
RPC	Remote Procedure Call
RTF	Rich Text Format
RTM	Requirements Traceability Matrix
SQL	Structured Query Language
SSP	specifikace softwarových požadavků
SVG	Scalable Vector Graphics
TCP/IP	Transmission Control Protocol/Internet Protocol
tzv.	takzvaný
UI	uživatelksé rozhrnaí
UML	Unifikovaný modelovací jazyk
URL	Uniform Resource Locator
VML	Vector Markup Language
WYSIWYG	„What you see is what you get“
XHR	XMLHttpRequest
XHTML	Extensible HyperText Markup Language
XLS	MS Excel
XML	Extensible Markup Language

Seznam obrázků

Obrázek 1: Schéma vodopádového modelu (Zdroj: Vondrák, 2002) 8
Obrázek 2: Struktura jazyka UML (Zdroj: Arlow & Neustadt, 2007) 10
Obrázek 3: Stavební bloky jazyka UML (Zdroj: Arlow & Neustadt, 2007, s. 35)..... 10
Obrázek 4: Diagramy UML(Zdroj: Arlow & Neustadt, 2007, s. 37) 12
Obrázek 5: Společné mechanismy UML (Zdroj: Arlow & Neustadt, 2007) 13
Obrázek 6: Kategorie modulů Spring Frameworku (Zdroj: Walls, 2013, s 22) 22
Obrázek 7: Role Hibernate v Java aplikaci (Zdroj: Bauer & King, 2015) 23
Obrázek 8: Proces GWT kompilátoru (Zdroj: Tacy, Hanson, Essington, & Tökke, 2013) 25
Obrázek 9: Sekvenční digram ukazující požadavek uživatele na spuštění GWT aplikace (Zdroj: Tacy, Hanson, Essington, & Tökke, 2013)..... 26
Obrázek 10: Implementace RPC GWT do aplikace (Zdroj: Tacy, Hanson, Essington, & Tökke, 2013) .. 27
Obrázek 11:Use case digram - Modul tiskových výstupů (Zdroj: vlastní zpracování) 38
Obrázek 12: Architektura aplikace (Zdroj: vlastní zpracování)..... 40
Obrázek 13: Princip DAO (Zdroj: Core J2EE Patterns – Data Access Object, 2015)..... 41
Obrázek 14: Struktura aplikace (Zdroj: vlastní zpracování)..... 42
Obrázek 15: ER diagram (Zdroj: vlastní zpracování)..... 48
Obrázek 16:Diagram tříd DTO (zdroj: vlastní zpracování)..... 49
Obrázek 17: Grafický návrh uživatelského rozhraní (Zdroj: vlastní zpracování) 50
Obrázek 18: Uživatelské rozhraní pro správu tiskových šablon (Zdroj: vlastní zpracování) 51
Obrázek 19: Sekvenční diagram ukládání šablony (Zdroj: vlastní zpracování)..... 51
Obrázek 20: Sekvenční diagram načtení šablony (Zdroj: vlastní zpracování) 52
Obrázek 21: Panel pro správu dat (Zdroj: vlastní zpracování)..... 52
Obrázek 22: Uživatelské rozhraní pro správu dat (Zdroj: vlastní zpracování)..... 53
Obrázek 23: Stromová struktura reportu (Zdroj: vlastní zpracování) 53
Obrázek 24: Kreslící plátno webové aplikace..... 54
Obrázek 25: Formulář formát stránky (Zdroj: vlastní zpracování) 55
Obrázek 26:AccordionLayoutContainer - Kontrolor, Elementy, Nástroje (Zdroj: vlastní zpracování) .. 56
Obrázek 27: Diagram tříd elementů (Zdroj: vlastní zpracování) 57
Obrázek 28: Chybová konzole (Zdroj: vlastní zpracování)..... 57
Obrázek 29: Uživatelské rozhraní správa obrázku (Zdroj: vlastní zpracování)..... 58
Obrázek 30: Sekvenční diagram pro generování tiskového výstupu 58

Seznam tabulek

Tabulka 1: Hodnoty atributu priorita (Zdroj: Arlow & Neustadt, 2007) 31
Tabulka 2: Shrnutí požadavků na aplikaci (Zdroj: vlastní zpracování) 33
Tabulka 3: Jednotlivé případy užití (zdroj: vlastní zpracování)..... 36
Tabulka 4: RTM matice (Zdroj: vlastní zpracování)..... 37

Prohlášení o využití výsledků diplomové práce

Prohlašuji, že

- Jsem byl seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, diplomovou práci užít (§ 35 odst. 3);
- souhlasím s tím, že diplomová práce bude v elektronické podobě archivována v Ústřední knihovně VŠB-TUO a jeden výtisk bude uložen u vedoucího diplomové práce. Souhlasím s tím, že bibliografické údaje o diplomové práci budou zveřejněny v informačním systému VŠB-TUO;
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;
- bylo sjednáno, že užít své dílo, diplomovou práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne 14.7.2015

……

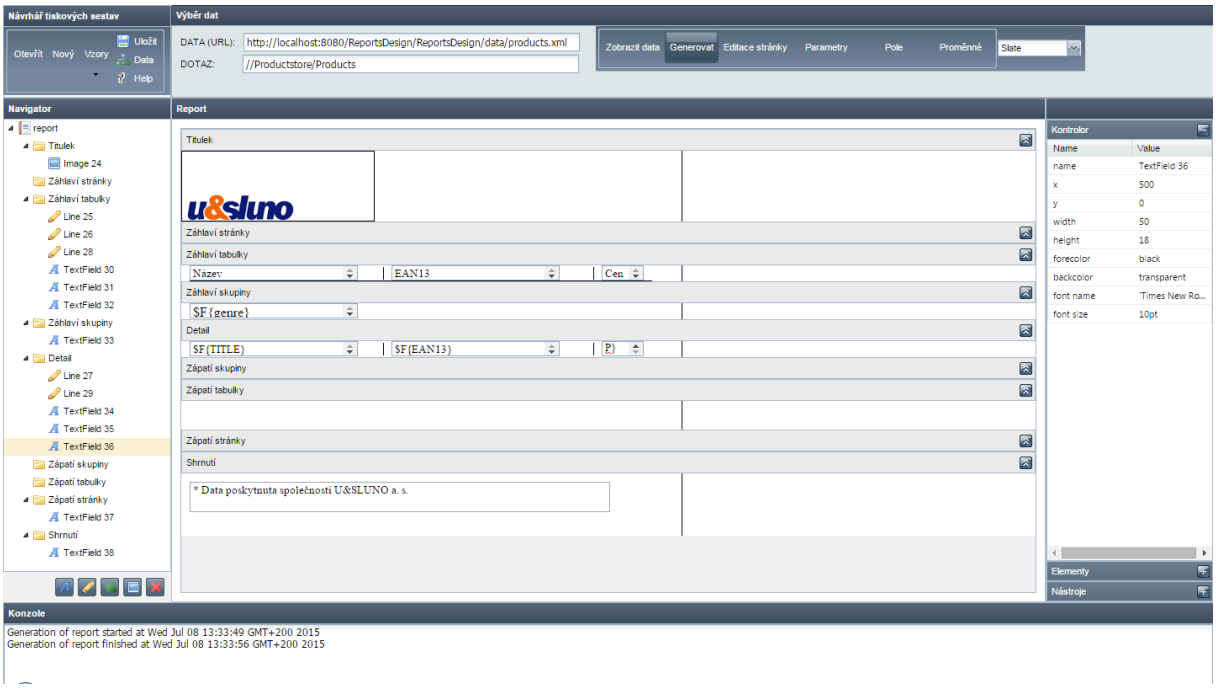
Bc. Radim Mikula

Seznam příloh

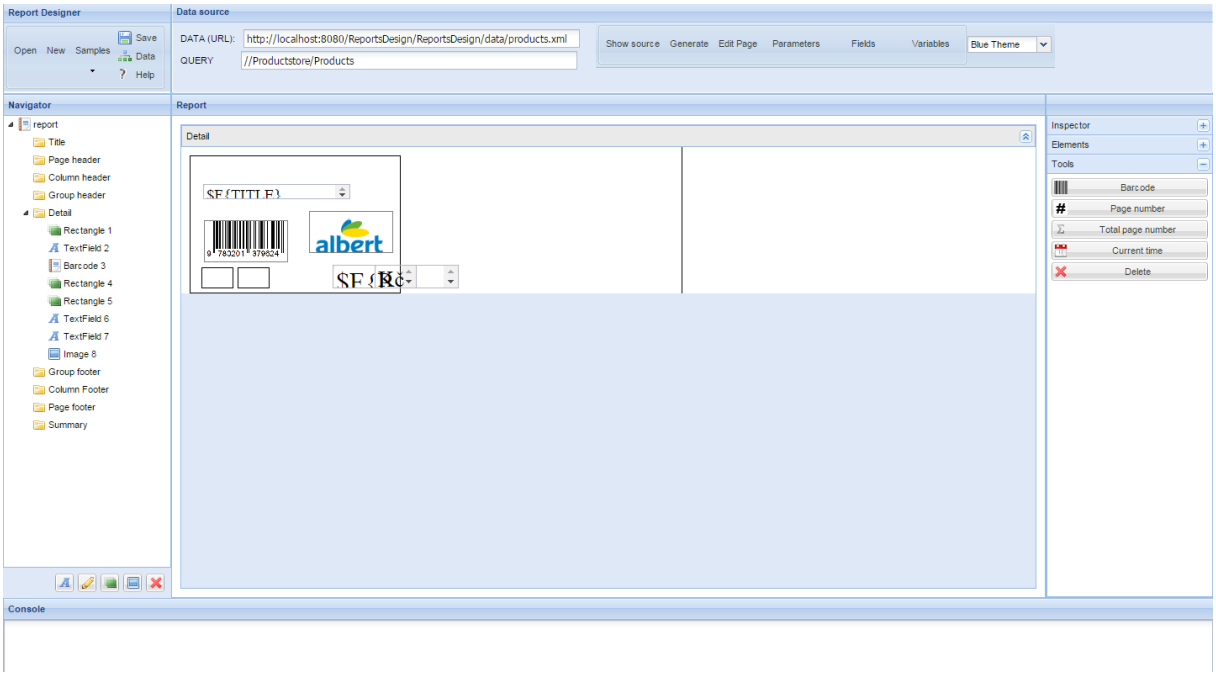
Součástí práce jsou tři přílohy:

1. Grafické návrhy aplikace.
2. Ukázka možných tiskových výstupů
3. Kompaktní disk.

Příloha č. 1 : Grafické návrhy aplikace



Obrázek: Grafický návrh aplikace (SLATE)



Obrázek: Grafický návrh aplikace (BLUE)

Příloha č. 2: Ukázka možných tiskových výstupů

u&sluno

Název	EAN13	Cena
MLÉKO ČERST.PET OLMA 1.5% 1L	22435118	15.40
MAKOVKA 60G	25322576	3.80
KAISERKA CEREÁINÍ 56G	24781114	2.09
BAGETA PŘ..TM.MAL. 45G	20671341	16.60
TAÂŠKA S UCHEM SLUNO-NOVÝ MOTI	24628501	1.70
TATRANKY CELOM. LÍSKOOŘ. 30G	25216714	9.30
BANÁNY PREMIUM	20443412	20.70
JABLKA ČERVENÁ	20902216	10.40
DALAMÁNEK PR. 95G	24413282	4.00
CIBULE LAHŮDKOVÁ SVAZEK KS	20446789	4.80
TATRANKY CELOM. LÍSKOOŘ. 30G	25216714	9.40
POKLADNÍ TAŠKA SLUNO QUAL.	24968881	0.50
POKLADNÍ TAŠKA TAÂŠKY SLUNO	24498128	1.70
ROHLÍK 43GR	20480905	1.50
TYČ.ČOK.KINDER BUENO OŘ.43G	20318918	8.90
VODA PERLIVÁ 1.5L	23331563	6.70
ROHLÍK SOJOVÝ 60GR	21471971	3.40
ROHLÍK SYPANÝ SÝREM 60G	21482922	5.50
EXPRESS ČOKO ROLKA 72G	24890656	4.20
EXPRESS ČOKO ROLKA 72G	24890656	4.20
ČERSTVÁ VEJCE M/10 KS	23617438	19.90
JABLKA ZELENÁ	25362671	14.70
KOBLIHA S DŽEMEM 50GR	20428990	4.80

* Data poskytnuta společnosti U&SLUNO a. s.

Obrázek: Tabulkový výstup

Obrázek: Výstup ve formě etiket

Příloha č. 3: Kompaktní disk

Obsah kompaktního disku:

1. Text diplomové práce.
2. Zdrojové kódy a zkompilovaná webové aplikace.
3. Základní struktura databáze ve formě SQL příkazů.